# METAHEURISTICS FOR $\mathbb{NP}$-HARD COMBINATORIAL OPTIMIZATION PROBLEMS

Dinh Trung Hoang

(B.Sc, National Uni. of Vietnam)

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2008

## ACKNOWLEDGEMENTS

I would like to extend my gratitude and deepest appreciation to Dr. A. A. Mamun for his inspiration, excellent guidance, endless support and encouragement during the work. He has always made himself available for discussion whenever I encountered problems with the project. His erudite knowledge and deepest insights have been the most inspiration and made this research work a rewarding experience. I owe an immense dept of gratitude to him for having given me the curiosity about metaheuristics. Without his kindest help, this thesis and many others would have been impossible.

Thanks also go to the faculties in Electrical & Computer Engineering Department in National University of Singapore, for their constant encouragement and valuable advice.

Acknowledgement is extended to National University of Singapore for awarding me the research scholarship and providing me the research facilities and challenging environment during my study time.

I sincerely acknowledge all the help from all members in Mechatronics & Automation lab, the National University of Singapore, in particular, my friends Dr. Tang K.Z., Mr. Trung, Mr. Zhu Zhen, Ms. Liu Jin, and Mr. Guan Feng for their kind assistance and friendship.

Last but not least, I would thank my family members for their support, understanding, patience and love during this process of my pursuit of a PhD., especially to my pretty and cunning sister Hang, my silly but handsome brother Hieu for all of their constant support for and sharing with me in whatever problems or happiness I faced or had since day one. During when struggling for preparing the oral defence, I was receiving strong support from

my beloved Mummy who had come to Singapore twice just to help me any single thing and attended my oral defence. I am very appreciated all of what she has done to me. Also I would thank to my girlfriend Ngoc Kim for her ongoing strong and eternal love gave to me while I was stressful with industrial work at TECH and still trying finalizing the last version of the thesis. This acknowledgement would not complete if the great sacrifice of my Dad for his children's further study was not recalled. This thesis, thereupon, is dedicated to them for their infinite stability margin.

METAHEURISTICS FOR NP-HARD COMBINATORIAL OPTIMIZATION PROBLEMS

Dinh Trung Hoang

National University of Singapore 2008

# Abstract

Combinatorial Optimization problems (COPs) are highly theoretical and of practical importance. Unfortunately, most of interesting COPs are proved to be intractable. Therefore, approximation approaches to those problems have received much intention since 1970s. During the past decades, a new kind of approximation algorithms, nowadays termed as *metaheuristic*, has emerged, providing a framework for solving many COPs by exploring the search space efficiently and exploiting the search history effectively.

Among approximation algorithms, metaheuristic algorithms are widely recognized as one of the most practical approaches for combinatorial optimization problems. Some noticeable representatives of metaheuristics are Simulated Annealing (SA), Tabu Search (TS), Evolutionary Computation (EC), Ant Colony Optimization (ACO) and so on. For many combinatorial optimization problems the established metaheuristic algorithms are considered to be the state-of-the-art methods. In this report, we present two parts of work. One is on Ant Colony Optimization; the other is on decomposition-based hybrid metaheuristics.

In particular, we propose a model of Ant algorithms that extends Graph-based Ant System (abbreviated as GBAS) model [106]. GBAS is the first and

most simple model which is used to study theoretical aspects related to convergence properties of ACO metaheuristics. All proposed to-date models for studying the convergence properties of ACO have not considered a widely-used technique which is to balance the exploration and exploitation process in almost all Ant-based algorithms. This technique is well-known in the research field of ACO and is called *pseudo-random proportional rule* or trade-off technique. To study the effectiveness of this technique in Ant-based algorithms from convergence perspective, an extended model of GBAS is proposed in one part of this report. Not only hold convergence properties as proved in GBAS, our model is also able to elucidate the practical role of this technique in Ant-based algorithms.

Inspired by findings from this extended model, we suggest and experiment with a time-dependent approach. This approach aims at practically improving performance of Ant-based algorithms through a adaptively-adjusting rule for the trade-off technique. To judge the effectiveness of this time-dependent approach, we integrate it into state-of-the-art Ant-based algorithms - which are Ant Colony System (ACS), Max-Min Ant System, Best-Worst Ant System- in two different scenarios: i) use local search procedures and ii) do not use local search procedure in any algorithm. By testing on some medium-scale benchmark instances of Traveling Salesman Problem, we show experimentally that the performance of the Ant-based algorithms employing the adaptively linear adjusting rule has been improved in comparison to that of the original Ant-based algorithms.

A field of research on hybridization of metaheuristics with basic techniques in Artificial Intelligence and/or Operations Research has emerged re-

cently and rapidly received attention of metaheuristics community. These hybrid metaheuristics aim at efficiently and effectively tackling large-scale real-world instances of COPs. Some findings in literature have suggested that the combination of classical artificial intelligence and operations research techniques with metaheuristics can be very beneficial for dealing with large-scale instances of some COPs. In the part of this report on hybrid metaheuristics, we present runtime analysis of a scheme of hybridization between metaheuristics and clustering (or decomposition) methods. In particular, we prove that decomposition-based search method formed by combining a decomposition technique with a problem-solving algorithm runs faster than methods that do not utilize decomposition techniques. The speedup gained, however, is bounded and the bounds can be computed in advance. The finding of such bounds has shed some light on theoretically elucidating the runtime efficiency of decomposition-based search algorithms over the non-decomposition-based ones. This is the first work using an unified but problem- and algorithm-independent framework to evaluate the effectiveness and efficiency of decomposition-based search algorithms in term of running time through the comparison to running time of alternative non-decomposition-based search algorithms.

Moreover, in that part of this report, we also address concerns over a disadvantage of decomposition-based methods, which relates to the failure of achieving optimal solutions in some scenarios. Those scenarios are simultaneously dependent on both problem-solving methods and structure of instances of optimization problems. Our finding suggests that given an inexact decomposition-based method for solving an optimization problem there

probably exist some instances of the problem for which the method fails to include any optimal solutions in the search space. This means no optimal solution can be found using such a method no matter how much time any algorithmic instance of the method is given to run.

To illustrate, we propose a simple inexact decomposition-based method to solve the Euclidean Traveling Salesman Problem (abbreviated as ETSP) and derive a sufficient condition on structure of ETSP instances such that if an instance of ETSP satisfies that condition, all of its optimal solutions will be contained into the search space generated by the proposed method; otherwise no optimal solution appears in that search space. However, the sufficient condition is applicable for a restricted number of subproblems, thus to make that condition more robust and applicable to large scale instances we extend it with additional assumptions on the structure of those large scale instances. The experimental results show that performance of a decomposition-based algorithm using ACS and derived from the sufficient condition is better than that of ACS on the same tests consisted of large scale clustered ETSP.

## TABLE OF CONTENTS

## LIST OF ALGORITHMS

## LIST OF TABLES

## LIST OF FIGURES

# Chapter 1

# Introduction

## 1.1 Background and Motivation

Combinatorial Optimization is a branch in applied mathematics, computer science and Operations Research. Most of problems studied in the early days of combinatorial optimization came from operations research, industrial management, logistics, engineering, computer science and military applications. But problems of this kind arise almost everywhere, and therefore combinatorial optimization has found successful applications in fields like archeology, biology, chemistry, economics, geography, linguistics, physics, sociology, and others. Combinatorial Optimization problems (COPs) are the main objectives of study in areas such as Computational Complexity Theory, Algorithmic Theory, and Artificial Intelligence. In fact, COPs are not only of academic interest but also of industrial interest. For example, a factory needs to determine the optimum order in which to manipulate resources. This is actually a variant of Production Scheduling Problem which is an instance of COPs. Another instance of COPs is the Protein Folding Problem (PFP) that appears in areas like Bioinformatics, Computational Chemistry, and in the pharmaceutical industry when one wants to know the spatial structure of a protein. Some of the related applications of COPs are Circuit Layout Design, Statistical Physics, Network Design, Transportation Science, and Computational Molecular Biology [104]. Due to the practical importance of COPs, numerous algorithms have been developed to solve them. These algorithms are classified as either exact or approximate algorithms based on how opti-

mal the solution found by the algorithm is . Exact algorithms are provably guaranteed to obtain optimal solutions to any instances of COPs in a given period of time [150]. Since most COPs are showed to be NP-hard, there is no deterministic polynomial time exact algorithm for those intractable COPs unless $\mathcal{P} = \mathcal{NP}$. Thus, the approximate algorithms, which provide an acceptable compromise between the quality of solutions and the runtime, have received much attention in the last few decades. This study focuses on a particular class of approximation algorithms, named *metaheuristics*, for solving COPs. An introduction of formal definitions of COPs can be found in the next section.

## 1.1.1 Combinatorial Optimization

### 1.1.1.1 The Optimization Problem

Optimization problems are generally formulated as follows:

**Definition 1.1.1.** *Optimization problem*

$$\begin{aligned} &\textit{minimize} && f(x) \\ &\textit{subject to} && x \in F. \end{aligned} \tag{1.1.1}$$

We call $f$ the *objective function*, $F$ the *feasible region* that satisfies all the given constraints, and a solution $x \in F$ a *feasible solution*. An element $x^* \in F$ such that

$$f(x^*) < f(x) \quad \text{for all } x \in F$$

is called an *optimal solution* to the problem. A problem which has $F \neq \emptyset$ is said to be *compatible* otherwise it is *incompatible*. A problem which has solution is said to be *solvable*. A solvable problem is necessarily compatible [151]. The

subsequent subsection will give more formal definitions for combinatorial optimization problems.

#### 1.1.1.2 Combinatorial Optimization

If $F$ has combinatorial features, for example *combination* or *permutation*, then the problem as in definition (1.1.1) is called a *combinatorial optimization problem*. One of the most general and formal definitions in of combinatorial optimization problem literature is given as follows.

**Definition 1.1.2.** *Given a finite number of objects, say $\Im$, and an "objective" function*

$$f : \Im \to S$$

*(where $S$ is an ordered set) which associates with every object a 'cost', 'value', 'weight', 'distance' or the like. Find an element of $\Im$ whose cost is minimum or maximum with respect to some criterion.*

There are numerous combinatorial optimization problems found in literature, for which computing exact optimal solutions is practically computationally intractable; e.g., those known as *NP*-hard [83], or polynomial-time but not practical. Those combinatorial optimization problems will be of interest to metaheuristics or decomposition-based methods.

It appears that many of the combinatorial optimization problems occurring in practice can be put in the following form which is more specific than the COPs definition (1.1.2).

**Definition 1.1.3.** *Let $F$ be a finite set and $\Im$ a set of subsets of $F$ called the set of feasible solutions. Let $c : F \to \mathbb{R}$ be a linear objective function. Find a feasible set*

$I^* \in \Im$ *such that*

$$\sum_{e \in I^*} c(e) = max \left\{ \sum_{e \in I} c(e) \mid I \in \Im \right\}$$

Problem (1.1.3) is called a *linear objective combinatorial optimization* [124].

## 1.1.2 On the Computational Complexity of Algorithms and No Free Lunch Theorem

### 1.1.2.1 Computational Complexity of Algorithms and $P$ vs $NP$

When designing an algorithm, one is generally interested in improving its efficiency as much as possible, where the efficiency of an algorithm is typically measured in terms of its computation *time*[1]. Efficiency is such a critical factor in design of algorithms, especially it is used as a metric for classifying COPs: a problem is regarded as *well-solved* if an algorithm is available to solve *efficiently* all its instances. It is also a meaningful and accepted naming convention [71] to call an algorithm "*efficient*" if it runs in time polynomially in the size of the instance. There is no algorithm proved to solve Traveling Salesman Problem efficiently. The same holds for a large set of other relevant problems: notwithstanding the great efforts of research community, there is

[1]Obviously, the actual computation time of an algorithm absolutely depends on the speed and on the hardware and software architectures of the computer on which it runs. A measurement which is machine independent can be modeled based on the number of basic operations needed by some abstract computation model, for instance, a Turing machine [159]. However, in the framework of this thesis, an informal understanding of this concept will suffice.

still a *class of hard* problems for which no efficient algorithm is available. It has been theoretically proved that if an efficient algorithm were available to solve any element of this class of problems, there would exist efficient algorithms to solve the remaining elements of this class. This is the fact due to an interesting property that this class holds. That property says that each member of the class is *reducible* to the others through a polynomial-time mapping; i.e., each instance of any problem belonging to the class can be transformed into a corresponding instance of any of the other problems in polynomial time. As a result, either all problems of the class are efficiently solvable, or none of them is. To date, there is no formal claim of neither of the two alternatives. Using the concepts in terms of Turing machine [7, 159], we call $NP$ a class of problems that can be solved in polynomial time by a *nondeterministic* machine. A subset of $NP$ which is noted as $P$ is defined as the class of problems that can be solved in polynomial time by a *deterministic* machine. Obviously, $P \subset NP$. However, the question whether $P = NP$ remains as one of the most challenging problems for theorists for decades.

#### 1.1.2.2 The No Free Lunch Theorem - *a priory* equivalence of search algorithms

A corpus of important theoretical findings on optimization algorithms are under a suggestive name of *No Free Lunch* theorems (abbreviated as NFL) [205, 206]. These results concern the problem of optimization from a rather abstract point of view. In the original form, they do not refer to any of the combinatorial optimization problems or to any specific search algorithm. Indeed, NFL theorems are proved for abstract models of *optimization process*,

and only some theoretical results have recently been proposed, bridging the gap between the abstract framework of NFL theorems and the actual practice in combinatorial optimization or explaining them in the view of computational complexity theory [22, 117, 204].

A NFL theorem for an abstract model of a search process in the original form informally states as follows:

> ...all algorithms that search for an extremum of a cost function perform exactly the same, when averaged over all possible cost functions. If algorithm A outperforms algorithm B on some cost functions, then loosely speaking there must exist exactly as many other functions where B outperforms A... [205].

From the statement, it implies that no search algorithm outperforms others on all optimization problems. Another NFL theorem for performance of a search algorithm on different class of optimization problems says that

> ...for any algorithm, any elevated performance over one class of problems is offset by performance over another class... [206]

However, Igel & Toussaint [117] claimed that NFL theorems should not hold on the classes of combinatorial optimization that are of practical relevance. Although these notable and interesting findings, unfortunately, they failed to show practical examples of classes of COPs for which NFL does not hold. There are still instances of COPs, such as Traveling Salesman Problem, Timetabling, Quadratic Assignment, for which the question whether NFL theorems hold is still remaining open.

### 1.1.3 Exact versus approximate approaches

Numerous algorithms have been devised for solving COPs. These algorithms can be classified as either *approximate* or *exact*[2].

Exact algorithms are guaranteed to obtain optimal solutions to any instance of problems within a computation time that is long enough. However, due to the fact that many problems of interest are NP-hard, for examples Minimum Flow Shop Scheduling, Minimum Bin Packing, Minimum Dynamic Storage Allocation, Traveling Salesman Problem so on (for a very complete list of $NP$ problems, see [7]), no deterministic polynomial-time algorithm has so far been known to solve them efficiently unless $P = NP$. In consequence, for many problems of interest, the applicability of exact algorithms is only to the extent of small- and medium-scale instances. Whereas, approximate methods are not guaranteed to find optimal solutions but practically they are able to obtain good solutions or near-optimal solutions in a relatively short period of time.

An *approximation method*, also called a *heuristic method* or simply a *heuristic*, is a well-defined set of steps for quickly identifying a high-quality solution for a given problem, where a *solution* is a set of values for problem unknowns and "quality" is defined by a stated evaluation metric or criterion. Solutions are usually assumed to be *feasible*, i.e. it meets all problem constraints. The purpose of heuristic methods is to identify problem solutions where time is

[2]The classification of algorithms into exact, heuristics, and metaheuristics that we adopt in this thesis is simplistic. A more well-defined taxonomy of algorithms could be given ( [160]), however, the classification adopted in this thesis satisfactorily serves the purpose of this thesis.

more important than solution quality, or the knowledge of quality.

Some heuristic methods are associated with problems for which an optimal or exact solution exists but is difficult to be computed by an *exact* algorithm. Heuristics are often used to identify "good" approximate solutions to such problems because of its shorter running time than if an exact algorithm is used. Heuristics can also be embedded within exact algorithms to expedite the optimization process.

Heuristics can be straightforward or more complex. Straightforward algorithms tend to have well-defined termination rules, as with greedy and local-neighborhood-search methods, which stop at a local optimum. More complex algorithms may *not* have standard termination rules and typically search for improved solutions until an arbitrary stopping point is reached.

Obtaining very good solutions to large scale instances of COPs in a significantly reduced amount of time is an advantage of heuristic algorithms over exact algorithms. However, as a result of No Free Lunch theorem, no heuristic algorithm is superior to another for all COPs. There are always well-designed heuristic algorithms for a specific COP. But those well-designed algorithms turn out to either be inapplicable to or have unsatisfactory performance for other COPs. Thus, since the 1970's, a new kind of approximation algorithms has emerged which aims at combining heuristic methods into higher level frameworks in order to explore effectively and efficiently a search space and be able to apply to many COPs with a little modification of the frameworks. These algorithms are nowadays termed as metaheuristics. The class of metaheuristics includes - but is not restricted to - Ant Colony Optimization (ACO) [62, 64–66], Evolutionary Computation (EC) [8,

100, 116, 140, 141, 143, 169, 183, 201], Tabu Search (TS) [53, 84, 90, 93, 96], POP-MUSIC [196], Simulated Annealing (SA) [1, 75, 118, 123, 127, 176], and Artificial Immune System [48, 49, 198].

### 1.1.4 Motivation

Combinatorial optimization problems are intriguing because they are often easy to state but very difficult to solve. Many of them arising in applications are *NP*-hard, that is, it is strongly believed that they cannot be solved to optimality within polynomially bounded deterministic computation time. Hence, to practically solve large instances one often has to use approximation methods which return near-optimal solution in relatively short time.

There are key issues in the development of thesis that is needed to elucidate. Those are concepts, that are often faced in practical applications, of *extremely large* instances or *too short* runtime. Both terms "large" and "short" for the size of instances and runtime, respectively, are quite relative and subjective. There not exists a rigorous definition of the critical problem size so as to classify problem instances. However, concept of such a critical instance size can be derived based on how much difficult to solve the instance in terms of metrics "time" and "optimality". For an example of TSP, with the size less than 700-city the instances are considered "small"; above 10,000 "very large" while above 1500 "large" [3].

A heuristic algorithm is usually dedicated to solving a specific optimiza-

[3]Take note that this is a rough classification which is based on the current computational technology and size of benchmark testing instances from TSPLIB

tion problem. The more well-designed the problem-specific algorithm, the more effective it is to solve the problem. However, that well-designed algorithm cannot effectively solve other optimization problems which share some common features with the one that it is able to solve effectively. A trend is to design a framework that is able to solve a class of optimization problems by combining user given black-box procedures - usually heuristics themselves - in a hopefully efficient way [19]. Such a framework has been referred to as a *metaheuristic*. A *metaheuristics* is, as defined in [69], a set of algorithmic concepts that can be used to define heuristic methods applicable to a wide set of different problems. A thorough review about definitions and concepts of metaheuristics is given in chapter 2.

Metaheuristics are generally applied to optimization problems to which no problem-specific heuristic method is able to solve them satisfactorily or not practical to implement. Most commonly used metaheuristics are targeted at combinatorial optimization problems, but of course can handle any problem that can be recast in that form, such as boolean equations [171].

Among successful metaheuristics, the approach of Ant Colony Optimization (ACO), which is a metaheuristic inspired by the behavior of real ants in finding the shortest paths from their nets to a food source and used to solve discrete optimization problems, will be studied in a part of this work. Another approach to COPs which is to use decomposition or clustering techniques to "embed" them into other problem-solving methods so as to derive effective methods for solving large scale instances will also be studied as another part of the work. The motivation for studying these both approaches is explained in the following subsections.

### Ant Colony Optimization

There have been extensive number of empirical studies on ACO ( [61, 63, 79, 165, 188, 190]) since its earliest system called *Ant System* [61] was introduced in Dorigo's Ph.D thesis. The approach is then extended into a higher level framework to be defined as a metaheuristic in [64]. However, works on analyzing the model behaviors theoretically had not started until the first impressive theoretical study by Gutjahr [106]. Since then, there have been few theoretical works carried out mainly due to the natural complex of modeling that metaheuristic for theoretical study purpose. Those works were dedicated to analyzing one of the most interesting questions on whether this metaheuristic will eventually converge or at least probabilistically converge to optimal solutions. In the first model named Graph-based Ant System (GBAS) for static COPs [106], strong constraints on structures of optimization problems and on the way the model encodes solutions have limited the extent of applicability of GBAS. Those constraints were then partially and/or completely relaxed and examined in other extended models [107, 108] which have stronger convergent properties than GBAS[4]. The last model proposed in [108] completely relaxed the constraints embodied in GBAS and as proven the relaxation did not change convergent properties of GBAS. However, as pointed out in [189], since GBAS model (and its extended versions) has not modeled ACO-based implementations closely enough, it is less accurate to apply convergent results of GBAS to those implementations. By adopting Gutjahr's method of proofing, Stutzle & Dorigo [189] proved some convergent properties for a class of ACO-based algorithms which are regarded as

[4]partially relaxed in [107] while completely in [108].

variants of Max-Min Ant System - one of most successful ACO-based implementations for some COPs ( [190, 191]). Although having larger applicability extent than Gutjahr's findings, Stutzle & Dorigo's finding still has a limitation that is convergent properties for that class of ACO-based algorithms are as weak as that for a random search [108].

Despite practically playing an important role in improving solution quality, a so-called *trade-off technique* that is commonly used in many implemented ACO algorithms [44, 65, 81, 190] has not been studied in any theoretical models so far. In those Ant algorithms, the trade-off technique represents a strategy of balancing between exploration and exploitation of search process. The strategy uses a fixed positive value for a systematic parameter that is referred to as *exploiting parameter*. Since that strategy has not been found in models of previous theoretical works, results from those works will not be able to predict or explain the importance in practice of the technique. In addition, in all variants of Ant System, the value of the exploiting parameter is always kept constant in runtime. There is neither empirical nor theoretical study carried out to investigate any affect on the performance of Ant algorithms if one adaptively adjusts the value of the exploiting parameter.

**Decomposition-based Approach**

When the size of input instances become very large, given the constraints on hardware-related computational resources like CPU and memory, we will soon realize that using a "straightforward" algorithm to solve that instance is not practical in the sense that we do not have enough memory to store

the input data or if we use "swap memory" [5] to keep data then the overall performance of the algorithm will be degraded and that the computation time of existing non-decomposition-based methods is too long for practical interest. Decomposition-based methods naturally come to resolve the issue. Rather than solving the instance as a whole that can bring us into the above issues, decomposition-based methods will solve it partially by "breaking" it into parts and solve each of those parts parallel or serially. A solution to the original instance is then formed by combining solutions to those parts. Depending on the design of a decomposition-based method and characteristics of underlying problems, the combination procedure can be done when all or a few number of parts are completely solved. In principle, we can describe a decomposition-based algorithm into four following steps:

a) Decompose (or partition or cluster) the large size instance into parts that have smaller size than the original instance. Each part is therefore considered as a subproblem

b) Solve each of these parts separately.

c) Ensemble solutions to these parts into a solution to the original instance.

d) If the solution to original instance is not satisfied with the objective, one can come back to either step a) or step b) or even step c) to further improve that solution.

[5]The memory is resulted by using empty space on hard disk to stimulate the physical memory - RAM. Since the access time to hard disk is normally much slower than to RAM, using swap memory will slow down overall performance of the algorithm.

We can find that the first three steps are the basic steps of such a decomposition-based algorithm.

Numerous works on decomposition-based methods can be found in literature ranging from exact approaches applied in mathematical programming ( [50, 103, 160]) to heuristic or metaheuristic approaches ( [6, 121, 170, 195, 196]).

In exact decomposition-based approaches, one can predict that how much memory would be saved if using decomposition-based method in comparison to using a straightforward one[6]. But a question that remains unanswered is that how many subproblems such exact methods need to solve in order to reach the optimal solution to the original instance [160]. That question basically relates to the question on convergence of iterative approaches to optimal solutions. However, one can ask oneself that even if a decomposition-based algorithm converges to optimal solutions to large scale instances, will the number of solved subproblems or their total size affects the runtime of the decomposition-based algorithm significantly? To evaluate whether the affect is "positive" or "negative", one needs to compare performance of that decomposition-based algorithm with that of the straightforward algorithm with an assumption that we have infinite (or sufficient enough) memory resource when running the straightforward one.

For inexact decomposition-based approaches, the same remaining unanswered question as in exact decomposition-based ones can be found, espe-

[6]Sometimes, we use "straightforward method" along with a "decomposition-based method" to refer to an unique method that only uses a problem-solving procedure which is the same as that in the decomposition-based to solve the same problem.

cially for a recently developed metaheuristic named POPMUSIC [196]. Moreover, in the case that total size of all solved subproblems is equal to the size of the original instance, one needs to answer another question on the performance comparison between a large number of decomposition-based heuristics[7] or hybridized metaheuristics [6, 170, 178, 192, 195], whose number of solved subproblems is known before the end of the execution of the (inexact) decomposition-based algorithms, and the straightforward heuristics or metaheuristics. Approaches in previous works when analyzing runtime performance of decomposition-based methods and comparing that performance with straightforward ones are merely either empirical-based or very problem-dependent and/or algorithm-dependent. There has so far been no unified approach, which is both problem- and algorithm-independent and relies on the general framework of decomposition-based methods, to carry out that analysis and comparison.

Moreover, all inexact decomposition-based methods will return solutions that are not proven to be optimal. The consequence of breaking a large structure (of the original instance) into smaller ones can lead to the situation that the search space contains poor solutions causing the search process to find it more difficult in reaching the (near-) optimal solutions.

For exact decomposition-based algorithms, that situation will cause the problem-solving process to run longer due to spending more time solving more number of subproblems before obtaining an optimal solution(s). For

[7] In decomposition-based hybrid approaches, hybridization is taking place between a metaheuristic and a classical clustering technique from Artificial Intelligence or Operation Research.

inexact ones, however, there are two distinct cases: i) the search space of inexact algorithms contains at least one optimal solution; ii) the search space does not contain any optimal solution. The former case i) is similar to the situation that might happen to the exact decomposition-based algorithms, while the later case ii) is not. Factors posing the case ii) can be from special structure of input instances and/or the "breaking" and assembling process. And if that case takes place for an inexact decomposition-based algorithm, no matter how long the algorithm spends solving an input instance, that algorithm will never obtain any optimal solution. Hence, there is a necessity to address the problem of how to improve solution quality of those inexact methods in the way of increasing capability of reaching optimal solutions of search process through guaranteeing those solutions in the search space.

## 1.2 Aims and Scope

The main aim of the first part of the study which focuses on Ant Colony Optimization metaheuristic is to investigate convergent behaviors of an extended model of GBAS. This model extends GBAS by incorporating the trade-off technique that is widely used in practice into the original GBAS. To be able to model that technique into the extended model, one systematic parameter will additionally be introduced into GBAS. Fundamentally, that parameter is to model the exploiting parameter in ACO-based implementations, hence we use the term "exploiting parameter" when referring to that systematic parameter. Following the theoretical study on convergence properties of ACO-based algorithms with the presence of trade-off technique is an empirical study on affect of not keeping the value of the exploiting parameter

fixed over runtime. The empirical study is to examine as many ACO-based algorithms as possible including the most successful ones by far.

The following specific goals will be expected to achieve in this part of the study:

- ☞ To investigate the convergence properties of this extended model under the same strong constraints presented in original GBAS.
- ☞ To examine whether or not the performance of Ant Colony Optimization algorithms is improved under the introduction of a dynamically pseudo-random proportional state transition rule.

To achieve the first goal, a strategy similar to that used for GBAS in [106] will be adopted to investigate convergence properties of the extended model of GBAS. However, due to the presence of the exploiting parameter in the extended model, there must be some necessary modifications to the original strategy. For achieving the second goal, an approach of using a function to dynamically update the value of the exploiting parameter will be investigated. The experimental results are to compare performance between Ant-based algorithms with the dynamically linear updating rule and those without that rule.

Firstly, theoretical findings of this part can contribute a part to fulfil the generally emergent demand in analyzing performance of ACO-based algorithms due to the fact that there are insufficient theoretical works in the field dedicated to that demand. Particularly, those findings may contribute a better theoretical understanding about the behavior of ACO algorithms with the trade-off mechanism. It is necessary to gain insight into the practical

importance of this mechanism by theoretical results so as to possibly derive fine-tuned values of systematic parameters in ACO-based algorithms. Results from the empirical study on dynamically updating rule specifically for the exploiting parameter can set experimentally forth empirical relationship between this parameter and others. Since tuning a set of values of systematic parameters is naturally an art (for instances, see [18], Chapter 1 in [17]), it is possibly beneficial to further extend this dynamically-updating approach - which tunes values of one parameter - to a set of parameters. Thus one contribution of this empirical study is to shed light on developing more complicated methods of deterministically or probabilistically tuning systematic parameters.

The study on ACO will limit itself in examining the convergence properties of the extended model of the original GBAS [106] in the event of the trade-off technique introduced. Thus, to examine whether or not a GBAS-extended model - which embodies this technique and relaxes strong constraints on structure of problems and solution encoding methods - converges is out of the scope of this study. Also, the study on the empirical part is to carry out investigation on the affect of dynamically updating rule using a linear updating function, hence investigating on such affect using any non-linear updating functions will not be in consideration.

The first aim of the second part of study on decomposition-based approaches is to examine runtime efficiency of these approaches in comparison to the non-decomposition-based ones as long as both of them use the same problem-solving algorithms. Comparing the runtime of a method with another is basically to examine the ratio of runtime of one method to that

of another. The strategy used to study that ratio is to evaluate its upper and lower bounds. The tighter the bounds, the better the evaluation. As we know that the "actual" amount of runtime of a certain implementation of any algorithm greatly varies and depends on hardware and software platform, on particular input instances, and on values of systematic parameters. To avoid bringing such sharp contrasts into the performance analysis of algorithms, we consider all input instances of a given size $n$ together. With this consideration, we express the runtime as a function of the input length of problem instances so that the performance analysis does not involve these contrasts. By modelling the runtime of such implementations as a function of the input length of problem instances, we aim at answering following basic questions:

- ☞ What is the bound(s) of the ratio of runtime of a decomposition-based method to runtime of a straightforward method when both of them are applied to the same instance?
- ☞ How tight is the bound(s)?
- ☞ How does the bound(s) change when the size of input instances is varied asymptotically[8]?

Findings from analysis of runtime performance for decomposition-based methods can contribute to better understanding on the pros and cons of those methods in terms of runtime. If the bounds of the ratio show that decomposition-based methods run faster than the straightforward ones, then they also help to show the limitation quantity above that former methods cannot run faster. Additional benefit of the analysis may be a guideline for

[8] i.e. The size approaches to infinity.

decomposition-based algorithms in which the number of subproblems being solved is unknown until the end of the execution of those algorithms.

The guideline possibly shows the expected number of such subproblems such that if dealing with more than that number of subproblems, a decomposition-based algorithm is likely to run slower than its corresponding straightforward algorithm (refer to section 4.2.2.4 for details).

This analysis is to focus on aspects related to runtime efficiency of decomposition-based methods in comparison to those straightforward methods. Thus, study on comparing solution quality of the former with the later is not in the boundary of this part of study. Also, we try to obtain bounds of that ratio as tight as good for as many cases as possible, however, deriving a method to show the tightest bounds for the most generic situation does not stay in the scope of the analysis.

As mentioned in the previous subsection 1.1.4, there is a concern about the quality of solutions of inexact decomposition-based methods due to the possible consequence of breaking the large structure of an instance into smaller structures (of so-called subproblems). The second aim of this part of the study is to address the concern over solution quality of inexact decomposition-based methods. In particular, that concern was addressed for the case in which the search space does not contain any optimal solution. The strategy is to use a typical *NP*-hard COP named Euclidean Traveling Salesman Problem (ETSP) with a simple decomposition-based method (able to solve ETSP) as the objects of this illustrative study. Our approach to the underlying concern is to point out constraints on structure of TSP instances such that if structure of the instances satisfies the pointed constraints then the search space of the

decomposition-based method will definitely contain all optimal solutions.

The contribution of this part of the study on solution quality of inexact decomposition-based methods is for the first time to propose a new view to the problem of improving their solution quality through proving sufficient conditions on structures of input instances such that those satisfied instances' optimal solutions will be included in search space of a given decomposition-based algorithm. One finding from this part is to highlight the point that even using the same inexact decomposition-based algorithm for a certain optimization problem, the search space may contain optimal solutions for a set of instances or may not for other set of instances. Equivalently, staying in search space (of those solutions) may be independent of features of certain decomposition-based methods while possibly dependent on structure of instances.

The scope of this part of study is to demonstrate a new view on the problem that inexact decomposition-based methods may be faced. The problem is related to the scenario that no optimal solution belongs to search space of the methods. Thus exact decomposition-based methods are definitely not the object of the study. Moreover, to serve the purpose of the demonstration, an intuitive yet simple example which uses ETSP and a simple decomposition-based algorithm to solve ETSP is employed for the study. Therefore, a thorough analysis on more complex cases like for other optimization problems with established decomposition-based algorithms goes beyond the scope of this work. Although narrowing the scope of study on using such simple example, we are still able to gain the aims of addressing the concern and proposing a new view on resolving the concern.

# Chapter 2

# Literature Review

This thesis provides not only contributions to theoretical study as well as practical applicability of Ant Colony Optimization on small- and medium-scale instances of COPs which will be presented in chapter 3, but also contributions to studies of decomposition-based algorithms on large scale instances which will then be discussed in chapter 4. Recently developed hybrid metaheuristics using classical clustering methods in artificial intelligence and/or operation research can be considered as a specific class of decomposition-based algorithms. Because the nowadays best-performing metaheuristic applications are composed of algorithmic components from different metaheuristics. Therefore, the study of a certain metaheuristic as well as the development of well-working applications of that metaheuristic require knowledge about the whole field of metaheuristics. It is important to remain open-minded towards other fields of metaheuristic research. For that reason we give a survey of the nowadays most important metaheuristics in this chapter.

In the first section of this chapter, definitions and taxonomy of metaheuristics are presented. Then, important characteristics of a metaheuristic, which are diversification and intensification, are technically explained at the end of the first section. Section 2.2 gives basic descriptions of state-of-the-art metaheuristics.

Section 2.3 is devoted to review recent works that, generally, aimed at improving performance of well-established metaheuristics. Those works consist of hybridization of metaheuristics with classical Artificial Intelligence

and/or Operations Research routines or using group theory and *landscape theory* to characterize and construct efficient and useful local structures of solution space of optimization problems.

Finally, the last section will give a summary of the chapter.

## 2.1 Metaheuristics - concepts, classification and characteristics

### 2.1.1 What is a Metaheuristic ?

In the 70ies, a new kind of approximate algorithm has emerged which basically tries to combine basic heuristic methods in higher level frameworks aimed at efficiently and effectively exploring a search space. These methods are nowadays commonly called *metaheuristics*. The term *metaheuristic* first introduced in [89] derives from the composition of two words. *Heuristic* means "to find" while the suffix *meta* means "beyond or at a higher level". Before this term was widely adopted, metaheuristics were often called *modern heuristics* [168]. This class of algorithms includes (in alphabetical order ) - but is not restricted to - ant colony optimization (ACO), evolutionary computation (EC), iterated local search (ILS), simulated annealing (SA), and tabu search (TS). So far, there is no widely accepted definition for the term metaheuristic. We list some of definitions in the following:

> A metaheuristic is formally defined as an iterative generation process which guides a subordinate heuristic by combing intelligently different concepts for exploring and exploiting the search

space, learning strategies are used to structure information in order to find efficiently near-optimal solutions [156].

A metaheuristic is an iterative master process that guides and modifies the operations of subordinate heuristics to efficiently produce high-quality solutions. It may manipulate a complete ( or incomplete) single solution or a collection of solutions at each iteration. the subordinate heuristics may be high (or love) level procedures, or a simple local search, or just a construction method [202].

Metaheuristics are typically high-level strategies which guide an underlying, more problem specific heuristic, to increase their performance. The main goal is to avoid the disadvantages of iterative improvement and, in particular, multiple descent by allowing the local search to escape from local minima. This is achieved by either allowing worsening moves or generating new starting solutions for the local search in a more ”intelligent” way than just proving random initial solutions. Many of the methods can be interpreted as introducing a bias such that high quality solutions are produced quickly. This bias can be of various forms and can be cast as descent bias (based on the objective function), memory bias (based on previously made decisions) or experience bias (based on prior performance). Many of the metaheuristic approaches rely on probabilistic decisions made dung the search. But, the main difference to pure random search is that in metaheuristic algorithms randomness is not used blindly but in an intelligent and biased form [187].

> A metaheuristic is a set of concepts that can be used to define heuristic methods that can be applied to a wide set of different problems. In other words, a metaheuristic methods that can be seen as a general algorithmic framework which can be applied to different optimization problems which relatively few modifications to make them adapted to a specific problem Metaheuristics Network at [68].

The definitions of metaheuristics as given above allow us to extract some fundamental properties by which metaheuristics are characterized:

- Metaheuristics are strategies that ”guide” the search process.
- The goal is to efficiently explore the search space in order to find (near)-optimal solutions.
- Techniques which constitute metaheuristic algorithms range from simple local search procedures to complex learning processes.
- Metaheuristic algorithms are approximate and usually non-deterministic.
- They may incorporate mechanisms to avoid getting trapped in confined areas of search space.
- The basic concepts of metaheuristics can be described on an abstract level (i.e., not tied to a specific problem).
- Metaheuristics are not problem-specific.
- Metaheuristics may make use of domain-specific knowledge in the form of heuristics that are controlled by the upper level strategy.

- Recently more advanced metaheuristics use search experience (embodied in some form of memory) to guide the search.

In the next subsection, a taxonomy of metaheuristics is presented in terms of characteristics such as nature-inspired, population-based, objective function type, the number of neighborhood structures, memory-based or memoryless, iteratively or constructively building new solutions, the number of cycles in termination (single run or repetitive), and instanced-based or model-based.

### 2.1.2 Classification of Metaheuristics

There are different ways to distinguish and describe metaheuristics. Depending on the features selected to differentiate among them, several classifications are possible, each of them being a result of a specific viewpoint. In this part, we briefly summarize the most important way of classifying metaheuristics as outlined in [106, 187].

**Nature-inspired vs. non-nature inspired.** One of the most intuitive ways to classify metaheuristics refers to their origins. Nature-inspired algorithms are, for examples, evolutionary computation, ant colony optimization, while examples of their counterpart are tabu search, partial optimization metaheuristic under special intensification conditions (POPMUSIC) (see subsection 4.1.2.1 on page 133 for more details). An algorithm is classified as nature-inspired if its basic idea is "borrowed" and modelized from nature's "phenomenon" instead of from the intrinsic of the problem. However, distinguishing metaheuristics in terms

of this classification appears difficult with the appearance of many recent hybrid algorithms that do not fit either class (or, in a sense they fit both!). Additionally, the difficulty comes from the hardness of attributing an algorithm to one of the two classes, for example the use of memory in tabu search may be wondered if it is nature-inspired or not.

**Single-point vs. population-based search.** Another characteristic that can be used to distinguish among metaheuristics is the number of current solutions used to determine the next state of system. If at any time only one solution is used so, then the algorithm is regarded to as a single-point-based search method. Otherwise, it is regarded to as a population-based search method. In addition, single-point-based methods are referred to as *trajectory methods* in this thesis. Instances of this class of metaheuristics are based on local search, such as, tabu search, iterated local search, guided local search, and variable neighborhood search. They all share a common characteristic that describes a trajectory in search space when they carry out the search. On the contrary, population-based methods, either perform search processes which can be considered as the "evolution" of a set of points in the search space, or they perform search processes which can be described as the evolutionary of a probability distribution over the search space (as, for example, estimation of distribution algorithm).

**Dynamic vs. static objective function.** Metaheuristics can also be differentiated according to characteristics of their objective functions. If an algorithm does not change its objective function during its execution,

it belongs to the class of metaheuristics whose objective functions are static. Whereas, other metaheuristics modify their objective functions during the search, for example, guided local search [203]. A positive consequence of modifying the objective function during the search is the possibility of escaping from local optima. Moreover, useful information collected during the search process can be used to alter the objective function.

**One vs. various neighborhood structures.** Numerous metaheuristics use only one certain neighborhood structure for the whole runtime, i.e. the landscape topology does not change during the execution of their implementations. Whereas, other metaheuristics, for example variable neighborhood search (VNS) (see subsection 2.2.2.1.2 on page 56 for more details about VNS) use a set of neighborhood structures in order to make the search more diversity by interchanging between different search landscapes.

**Memory-based vs. memory-less methods.** An important factor to distinguish metaheuristics is the way they make use of the search history, that is, whether or not they use a memory to remember the history. Since memory-less methods use only available information of the current state of the search process to decide the next action, their operations are regarded to as a Makov process. Examples of memory-less algorithms are variable neighborhood search (VNS) and greedy randomized adaptive search procedure (GRASP) (see subsection 2.2.2.1.1 on page 54). To memory-based approaches, there is a differentiation between the use

of short- and long-term memory. While the short-term memory keeps track of only recently visited solutions or performed moves, the long-term one is used to store accumulation of synthetic parameters about the search. Nowadays, using memory becomes a fundamental feature of efficient and effective metaheuristics for examples ant colony optimization (ACO)[1], tabu search (TS) (see subsection 2.2.2.3 on page 61).

**Iterative vs. constructive.** Metaheuristics can be classified based on the way they build up a new solution. An *iterative* heuristic starts with a (set of) complete feasible solution(s) and then obtain a new (set of) solution(s) by changing this (these) solution(s) in an iterative process so as to enhance quality of the current solution(s), whereas a *constructive* heuristic builds up a new (set of) solution(s) "from scratch" by adding opportunely defined solution components to an initially empty partial solution. Components adding is done until a solution is complete or other stopping criteria are met. A prominent example of iterative heuristics is Local Search (LS for short) (see section 2.2.2.1 on page 51 for more details about TS), where a new solution is resulted from changes of the current solution based on a so-called neighborhood structure, and that of constructive heuristics is Greedy Heuristics (GH for short) [30] where the final solution is gradually built up in a linear process controlled by "gains" of components which are accepted to be inserted at a certain step.

[1]See subsection 2.2.1.4 on page 2.2.1.4 for a general overview or chapter 3 on page 48 for a detailed review.

**Single run vs. repetitive** This classification is based on the way that heuristic procedures terminate *Single-run* procedures stop as soon as a certain internal termination condition is met (for example, a local optimum has been reached in the case of LS, or the constructive process has finished in the case of GH). In *repetitive* procedures, the user dictates the amount of runtime he wants to spend and the quality of the solution enhanced as a function of the runtime. The single-run heuristic procedures are fast but result in moderate solution quality, whereas the repetitive heuristic procedures might obtain the desired quality of the solutions at the cost of very large computation time.

**Instance-based vs. Model-based search.** Metaheuristic search methods can also be classified as either *instance-based* or *model-based* similarly to what is done in the machine learning field [166]. A search method is regarded to as instanced-based one if it generates new candidate solutions by using solely the current solutions or a the current batch solutions (or also called current "population" of solutions). Most of classical search methods may belong to the instance-based class. Typical examples of this class are genetic algorithms [116], or trajectory-based search algorithms (local search), such as, for examples, simulated annealing [75, 138, 139], tabu search [89, 96], variable neighborhood search [113, 144]. In contrast to instance-based methods, during the last decade, several methods, which may be considered to belong to the model-based class, have been proposed. In model-based search approaches, set of candidate solutions is generated by employing a *parameterized probabilistic model* that is iteratively updated by "experience" (normally by previous obtained

solutions) in such a way that the search process will focus on the regions containing potentially high-quality solutions.

At a very general level, the model-based search approach solves an optimization problem by repeating the following two steps [69]:

- ☞ Candidate solutions are constructed using a parameterized probabilistic model, that is, a parameterized probability distribution over the solution space.
- ☞ Candidate solutions are evaluated and then used to modify the probabilistic model in a way that is deemed to bias future sampling toward low-cost solutions. Also note that, the model's structure may be fixed in advance, with solely the model's parameters being updated, or alternatively, the structure of the model may be allowed to change in runtime as well.

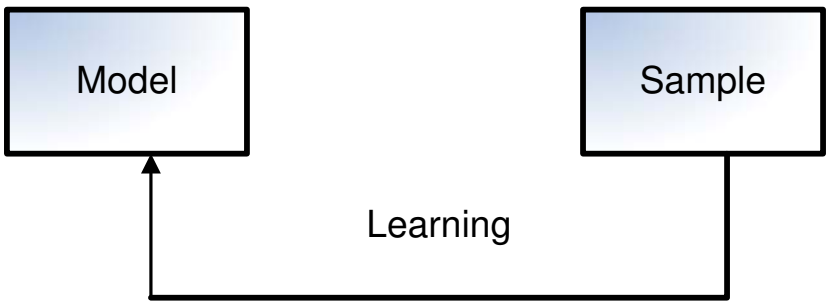


Figure 2.1: The basic scheme of a model-based search algorithm.

The concept of the term "model" in this classification should be cautiously awoken of. This term denotes an *adaptive stochastic mechanism* for generating candidate solutions and not an approximate represen-

tation of the environment as in, for example, reinforcement learning[2] [193]. Early works related to the model-based search approach, for examples, population-based incremental learning [9], ant colony optimization [39, 61, 65, 69, 106]. However, the first work that gave an explicit description of the model-based idea is credited to [21]. Figure (2.1)

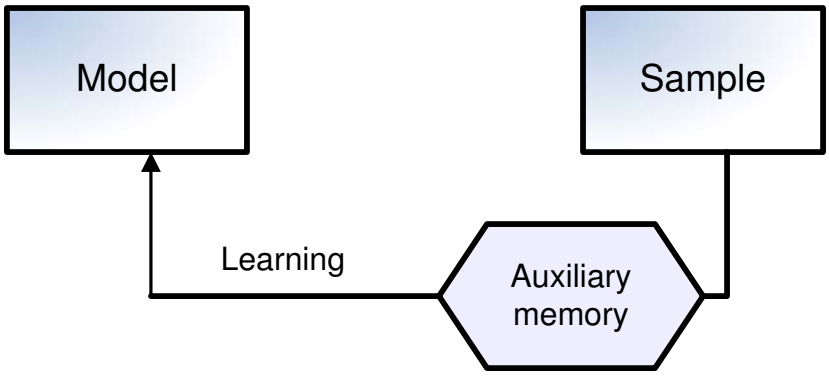


Figure 2.2: A model-based search algorithm with auxiliary memory.

shows a general schematic description of the model-based search approach. It is noteworthy that Fig. (2.1) presents the "purest" configuration of the model-based search approach. This configuration describes the approach whose model update rules are based on only the current solutions. Meanwhile, many model-based search algorithms additionally use an auxiliary memory as described in Fig. (2.2) to update their model. This auxiliary memory store information collected during the search process. A well-known realization of the model-based

[2]However, the term "model" in the model-based search methods for combinatorial optimization problems can have a close connection with that in reinforcement learning when that term is considered as an attempt to model the structure of the "promising" solutions.

approach using an auxiliary memory is the recently developed metaheuristic namely *estimation of distribution algorithm* (EDS) [148, 152]. See Fig. (2.3) for a schematic description of EDS in the view of a model-based metaheuristic.

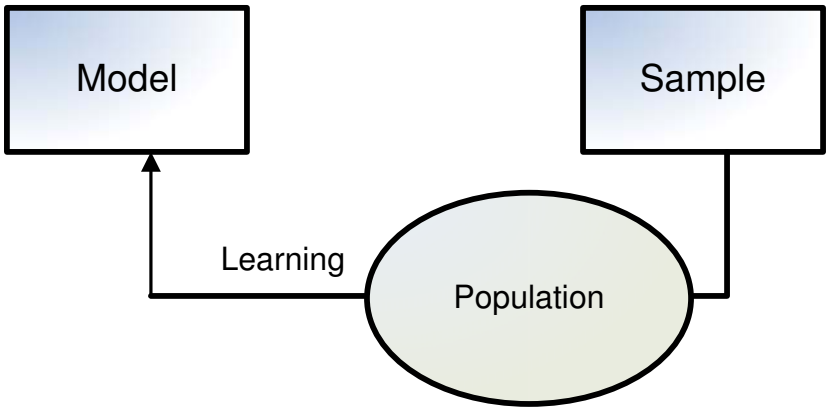


Figure 2.3: Schematic description of Estimation of Distribution Algorithm.

In the following subsection, we will present the concept of diversification and intensification that are two mainly powerful forces bringing high performance to metaheuristic applications.

### 2.1.3 Diversification and Intensification in Metaheuristics

*Effectively* and *efficiently* searching the solution space is the main objective that metaheuristics should be aimed at. A metaheuristic is actually efficient and effective if its search process is guided "smartly" enough such that not only intensively exploit areas with high-quality solutions but it also explores unvisited area when deemed necessary. These goals are nowadays conceptualized in the metaheuristic area as *intensification* and *diversification* (abbreviated

as *I&D*. These terms originated in TS field [96]. Related concepts have been popularly used in EC field and denoted by *exploitation* and *exploration* that are related to intensification and diversification respectively. But the concepts of exploitation and exploration have more restricted meaning. Indeed, they often refer to short-term search strategies tied to randomness, while diversification and intensification describe medium- and long-term search strategies based on the usage of memory. Since the numerous different methods of employing memory become increasingly important in the whole field of metaheuristics, the notions diversification and intensification are more and more adopted and understood in their original meaning.

A reference to an extensive and rather complete discussion about intensification and diversification is given in [19].

In the following paragraphs, some high-level descriptions of diversification and intensification found in literature are quoted.

> Two highly important components of tabu search are intensification and diversification strategies.Intensification strategies are based on modifying choice rules to encourage move combinations and solution features historically found good. They may also initiate a return to attractive regions to search them more thoroughly. Since elite solutions must be recorded in order to examine their immediate neighborhoods, explicit memory is closely related to the implementation of intensification strategies. The main difference between intensification and diversification is that during an intensification stage the search focuses on examining neighbors of elite solutions.[...]The diversification stage on the other hand

encourages the search process to examine unvisited regions and to generate solutions that differ in various significant ways from those seen before [96].

After a local minimizer is encountered, all points in its attraction basin lose any interest for optimization. The search should avoid wasting excessive computing time in a single basin and diversification should be activated. On the other hand, in the assumptions that neighborhoods have correlated cost function values, some effort should be spent in searching for better points located close to the most recently found local minimum point (intensification). The two requirements are conflicting and finding a proper balance of diversification and intensification is a crucial issue in heuristics [14].

A metaheuristic will be successful on a given optimization problem if it can provide a balance between the exploitation of the accumulated search experience and the exploration of the search space to identify regions with high quality solutions in a problem specific, near optimal way [187].

Intensification is to search carefully and intensively around good solutions found in the past search. Diversification, on the contrary, is to guide the search to unvisited regions. These terminologies are usually used to explain the basic elements of Tabu Search, but these are essential to all the metaheuristic algorithms. In other words, various metaheuristic ideas should be understood

from the view point of these two concepts, and metaheuristic algorithms should be designed so that intensification and diversification play balanced roles [208].

Holland frames adaption as a balance between *exploration* (the search for new, useful adaptations) and *exploitation* (the use and propagation of these adaptations) [116]. The tension comes about since any move toward exploration testing previously unseen schemata or schemata whose instances seen so far have low fitness takes away from the exploitation of tried and true schemas. In any system (e.g., a population of organisms) required to face environments with some degree of unpredictability, an optimal balance between exploration and exploitation must be found. The system has to keep trying out new possibilities (or else it could over-adapt and be inflexible in the face of novelty), but it also has to continually incorporate and use past experience as a guide for future behavior [143].

All these descriptions share the common view that there are two forces for which an appropriate trade-off has to be found. Moreover, intensification and diversification can be considered as effects of algorithm components. For the sake of convenience, define *I&D component* as any functional or algorithmic component that has a diversification and/or an intensification effect on the search process. Examples of I&D components are genetic operators, perturbation of probability distribution, changes in objective functions, and the usage of tabu lists. Thus, I&D components are operators, actions, or search strategies of metaheuristic algorithms. Although a still widely spread view

that there are algorithmic or functional components that have either a diversification or an intensification effect, there are many components that have both effects. In I&D components, that are normally labeled as diversification, the diversification component is stronger than the intensification component, and vice versa.

I&D components in metaheuristics can be divided in basic (or intrinsic) ones and strategic ones. The basic I&D components are defined by the basic ideas of a metaheuristic. On the contrary, the strategic I&D components are composed of techniques and strategies the algorithm designers purposely add into the basic metaheuristic in order to improve performance by incorporating medium- or long-term strategies. Most of these strategies are originally developed in the context of a specific metaheuristic. However, it becomes more and more apparent that many of these strategies can also be very useful when applied in other metaheuristics.

For example, the basic idea of TS is a neighbor choice rule using one or more tabu lists. This I&D component has two effects on the search process. The restriction of the set of possible neighbors in every step has a diversifying effect on the search, whereas the choice of the best neighbor in the restricted set of neighbors (the best non-tabu move) has an intensifying effect on the search. The balance between these two effects can be varied by the length of the tabu list. Shorter tabu lists result in a lower influence of the diversifying effect, whereas longer tabu lists result in an overall higher influence of the diversifying effect.

Another example is the following one. Ant Colony Optimization provides an I&D component that manages the update of the pheromone values. This

component has the effect of changing the probability distribution that is used to sample the search space. It is guided by the objective function (solution components found in better solutions than others are updated with a higher amount of pheromone) and it is also influenced by a function applying the pheromone evaporation. The effect of this mechanism is basically the intensification of the search, but there is also a diversifying component that depends on the greediness of the pheromone update (the less greedy or deterministic, the higher is the diversifying effect).

The two following subsections - subsection 2.2.1 and subsection 2.2.2 - will be devoted to reviews of metaheuristics that mainly belong to population-based and trajectory classes of metaheuristics respectively.

## 2.2 Some state-of-the-art metaheuristics

In this section, we will present state-of-the-art metaheuristics that are classified as population-based and trajectory approaches. Population-based methods including Evolutionary computation (EC), scatter search and path relinking (SS and PR), estimation of distribution algorithms (EDAs), and ant colony optimization (ACO) are presented in the following subsection, while trajectory ones including greedy randomized adaptive search procedures (GRASP), variable neighborhood search (VNS), simulated annealing (SA), and tabu search (TS) are presented in the other subsection of this section.

### 2.2.1 Population-based Approaches

In this subsection, a number of state-of-the-art population-based metaheuristics will be presented. They are consisting of Evolutionary Computation, Scatter Search and Path Relinking, Estimation of Distribution Algorithms, and Ant Colony Optimization.

#### 2.2.1.1 Evolutionary Computation

*Evolutionary computation* (EC) algorithms are inspired by natural capability to evolve of living being so as to well adapt to their environment. EC algorithms can be characterized as computational models of evolutionary processes. At each iteration, a number of operators are applied to the individuals of the current population to generate the individuals of the population of the next generation (iteration). Usually, EC algorithms use operators called *recombination* or *crossover* to recombine two or more individuals to produce new individuals. They also use *mutation* or *modification* operators which cause a self-adaptation of individuals. The driving force in evolutionary algorithms is the *selection* of individuals based on their *fitness* (which can be based on the objective function, the result of a simulation experiment, or some other kinds of quality measure). This corresponds to the principle of *survival of the fittest* in natural evolution. It is the capability of nature to adapt itself to a changing environment, which gave the inspiration for EC algorithms.

There have been a variety of slightly different EC algorithms proposed over years. Basically, they fall into three different categories which have been developed independently of each other. These are evolutionary programming (EP) as introduced by Fogel in [77] and by Fogel et al. in [78],

evolutionary strategies (ES) proposed by Rechenberg in [167] and genetic algorithms (GAs) initiated by Holland in [116] ( [100, 143, 169, 201] for further references). EP arose from the desire to generate machine intelligence. While EP originally was proposed to operate on discrete representations of finite state machines, most of the present variants are used for continuous optimization problems. The latter also holds for the most present variants of ES, whereas GAs are mainly applied to solve CO problems in early dates but the trend is to devote GAs to optimization problems including those in continuous domain.

**Algorithm 1** Evolutionary Computation (EC)

```
P ← GenerateInitialPopulation()
Evaluate (P)
while termination conditions not met do
  P′ ← Recombine(P)
  P″ ← Mutate(P′)
  Evaluate(P″)
  P ← Select(P″, P)
end while
```

While there are variants of EC like steady-state and generational approaches ( [182]), the algorithm ( 1 ) shows the basic structure of EC algorithms. In this algorithm, $P$ denotes the population of individuals. A population of offspring is generated by the application of *recombination* and *mutation* operators and the individuals for the next population are *selected* from the union of the old population and the offspring population. There are many other ways to perform an EC simulation. For example, there are steady-state and generational approaches, selection operator being applied before recombination and mutation operators [].

EC algorithms have been applied to most CO problems and optimization

problems in general. Recent successes were obtained in the rapidly growing bioinformatics area ( [33]), but also in multi-objective optimization [32]. For an extensive collection of references to EC applications we refer to [8].

#### 2.2.1.2 Scatter Search and Path Relinking

The evolutionary approach called *scatter search* (SS), and its generalized form called *path relinking* (PR), originated from formulations for creating composite decision rules and surrogate constraints [95, 97, 128]. Recent studies demonstrate the practical advantages of these two approaches for solving a diverse array of problems from both classical and real world settings in both discrete and nonlinear optimization( [76]). For an extensive list of applications of SS and PR, refer to [98, 99] and Chap 19 in [63].

Scatter search and path relinking mainly contrast with other evolutionary procedures, such as genetic algorithms, by providing unifying principles for joining (or recombining) solutions based on generalized path constructions (in both Euclidean and neighborhood spaces) and by using strategic designs where other approaches resort to randomization [92]. Additional advantages are given by intensification and diversification mechanisms that exploit adaptive memory, drawing on foundations that link scatter search and path relinking to tabu search. Thus, scatter search is regarded to as a historical bridge between evolutionary procedures and the adaptive memory strategies of tabu search [91].

**2.2.1.2.1 Scatter Search** (SS) uses strategies for linearly combining solution vectors[3] that have been proved to be effectivein a variety of problem settings. SS, from the standpoint of metaheuristic classification, may be viewed as an evolutionary (population-based) algorithm that constructs solutions by combining others. It derives its foundations from formulations originally proposed for combining decision rules and surrogate constraints in the context of integer programming. The goal of this methodology is to enable the implementation of solution procedures that can derive new solutions from combined elements in order to yield better solutions than those procedures that base their combinations only on a set of original elements ( [94]).

A set of points so-called *reference points* that constitutes "good" solutions resulted from previous solution attempts is used in scatter search's operation. The "good" solutions are defined as those including special criteria, like diversity, that purposely go beyond the objective function value. By combining the reference points which correspond to population of individuals in EC algorithms, the approach generates new points each of which is mapped into an associated feasible point.

Basically, there are five methods consisted in SS that are explained as follows:

1. A *diversification generation method* to create a collection of diverse trial solutions (or seed solutions), using one or more arbitrary trial solutions as an input.

[3] Actually, SS uses linear combination of a population subset to create new solutions. A special operator is, thus, used to ensure their feasibility and to improve their quality.

2. An *improvement method* to transform a trial solution into one or more improved trial solutions. This method does not require the feasibility of its input and output, although normally output's feasibility is requested.

3. A *subset generation method* to operate on the reference set to produce a subset of its solutions as a basis for creating combined solutions. The most common subset generation method is to produce all pairs of reference solutions (i.e. all subsets of size 2). The role of this method in SS is similar to that of the operator "selection" in EC algorithms.

4. A *solution combination method* to transform a given subset of solutions produced by the Subset Generation Method into one or more combined solutions. This combination method is analogous to the crossover operator in genetic algorithms, however, it is capable to combine more than two solutions.

A basic scatter search procedure is illustrated in Algorithm (2). This procedure starts with the creation of an initial reference set of solutions, symbolized as $S_{ref}$, from the set of diverse solutions $S_{div}$[4] by ChooseReferenceSet. The diversification generation method is employed (and implemented in DiversificationGenerator) to build a large collection of diverse solutions . The set of best solutions found in the search process is stored in $S_{best}$.

In the main loop of the procedure, at each iteration the following steps are iteratively done in a number of cycles that is a parameter of the algorithm. In the first step in a cycle, the subset generation method coded by GenerateSubsets () chooses a set of subsets of solutions $S_{sub}$ from the set of diverse

[4]Empirically, the size of $S_{div}$ is 10 times of that of $S_{ref}$.

**Algorithm 2** Basic Scatter Search - SS

```
S_div ← DiversificationGenerator(S_seed);
S_div ← LocalSeachImprover(S_div);
S_ref ← ChooseReferenceSet(S_div);
S_best ← ChooseBestOf(S_div);
while termination conditions not met do
  while termination criteria for the inner loop not met do
    S_sub ← GenerateSubsets(S_ref);
    S_trial ← CombineSolutions(S_sub)
    S_enhc ← LocalSeachImprover(S_trial);
    S_ref ← UpdateReferenceSet(S_ref, S_enhc);
  end while
  S_best ← ChooseBestOf(S_ref);
  S_div ← DiversificationGeneration(S_best);
  S_ref ← ChooseReferenceSet(S_div);
end while
```

solutions $S_{div}$. There are at least 2 solutions in a subset. The second step implements the solution combination method CombineSolutions and generates a set of trial solutions $S_{trial}$ from these chosen subsets of solutions $S_{sub}$. These trial solutions will then be enhanced in the next step by a local search procedure LocalSeachImprover that results in a set of solutions named $S_{enhc}$. The final step of the cycle carries out an update method UpdateReferenceSet for the reference set regarding to the set of enhanced solutions $S_{enhc}$. After the complete of number of these cycles, the set of diverse solutions are updated with respect to the elite solutions from the latest updated reference set. After that, a new reference set is chosen from the set of the resulting diverse solutions. The procedure is terminated as early as the termination criteria of the main loop is satisfied.

**2.2.1.2.2 Path Relinking** (PR) was originally proposed in [93] as an approach to integrate intensification and diversification strategies in the context

of tabu search . This approach is to generate new solutions by exploring trajectories that connect high-quality solutions (or sometimes ), by starting from one of these solutions and generating a path in the neighborhood space that leads toward the other solutions.

By starting from one or more high-quality solutions (or also referred to as *elite* solutions), PR generates paths in the solution space that lead to other elite solutions. These paths are explored in the search for better solutions. To generates the paths, moves are chosen to introduce attributes in the current solution that are present in the elite guiding solution. This selection progress of PR may be viewed as a strategy that attempts to incorporate attributes of elite solutions by favoring these attributes in the selected moves.

Algorithm (3) illustrates a pseudo-code of the PR procedure that is applied to a pair of solutions: $x$ (initial solutions) and $y$ (target solutions). First, the procedure calculate the so-called *symmetric difference* $\delta(x, y)$ between the initial and target solutions, i.e. a set of moves needed to reach $y$ from $x$. At each step in the "while" loop, the procedure tries all moves $m \in \delta(z, y)$ from the current solution $z$ and select the one that results the minimum cost solution, i.e. the one minimizes $f(z \odot m)$ where $z \odot m$ is the solution obtained by applying move $m$ to the current solution $z$. And $z \odot m^\star$ becomes the current solution after the best move $m^\star$ is obtained. And the set of available moves is updated by removing $m^\star$ from $\delta(z, y)$. The best solution $x^\star$ is necessarily updated if the current solution is better than the best-so-far solution $x^\star$. The procedure terminates when the symmetric difference is empty, i.e. the target solution $y$ is reached.

To balance between computation time and solution quality, several alter-

native ways of creating the set of moves and of how to explore the path have been considered [172]:

- *Periodical relinking:* PR is applied but instead only periodically.
- *Forward relinking:* PR is applied using the worst between $x$ and $y$ as the initial solutions and the other one as the target solution.
- *Backward relinking:* the roles of $x$ and $y$ is interchanged. PR is applied but using the best between $x$ and $y$ as the initial solutions and the other one as the target solution.
- *Back and forward relinking:* combines forward relinking and backward relinking together, the two different trajectories are explored.
- *Mixed relinking:* two path are simultaneously explored each from one solution until they meet at an intermediary solutions equidistant from $x$ and $y$.
- *Truncated relinking:* the full trajectory between $x$ and $y$ is not examined, but a part of it.

Scatter Search has received increasing interest in recent years. Among other problems SS is applied to are quadratic assignment problem [47], maximum clique problem [27], resource constrained project scheduling [199], graph coloring problem [110]. Path relinking is usually used as a component of other metaheuristics for examples Tabu Search [13, 115], and GRASP [172].

**Algorithm 3** Path Relinking - PR

**Input:** Initial solution $x$ and target solution $y$
**Output:** Best solutions $x^\star$ on a path linking $x$ and $y$
$\delta(x, y) \leftarrow$ CalculateSymetricDifference$(x, y)\{\delta(x, y)$ is a set of moves needed to reach $y$ from $x\}$
$f^\star \leftarrow \min\{f(x), f(y)\}$;
$x^\star \leftarrow \arg\min\{f(x), f(y)\}$;
$z \leftarrow x$;
**while** $\delta(z, y) \neq \emptyset$ **do**
  $m^\star \leftarrow \arg\min\{f(z \odot m) : \forall m \in \delta(z, y)\}$;
  $\delta(z \odot m^\star, y) \leftarrow \delta(z, y) \setminus \{m^\star\}$;
  $z \leftarrow z \odot m^\star$;
  **if** $f(z) < f^\star$ **then**
    $f^\star \leftarrow f(z)$;
    $x^\star \leftarrow z$;
  **end if**
**end while**

#### 2.2.1.3 Estimation of Distribution Algorithms - EDAs

In last decade more and more researchers tried to overcome the drawbacks of usual recombination operators of EC algorithms, which are likely to break good building blocks[5]. With this aim, a number of algorithms that are sometimes called *estimation of distribution algorithms* EDAs [148] have been developed (see Algorithm (4) for the algorithmic framework). These algorithms, which have a theoretical foundation in probability theory, are like EC algorithms based on population that evolve as the search processes. This new class of algorithms generalizes genetic algorithms by replacing the crossover and mutation operators with learning and sampling from the probability distribution of the best individuals of the population at each iteration of the

[5]Roughly speaking, a good building block is a subset of the set of solution components which result in a high average quality of all the solutions that can contain this subset.

algorithm ( [152]). Specifically, they work as follows. First, an initial population $P$ of solutions is randomly or heuristically generated. Then the following cycle is repeated until the termination criteria are met. A fraction of the best solutions of the current population (denoted by $P_{sel}$) are selected in function *ChooseFrom(P)*. Then from the solutions in $P_{sel}$, a probability distribution over the search space is derived in function *EstimateProbabilityDistribution($P_{sel}$)*. This probability distribution is then sampled in *SampleProbabilityDistribution(*$\mathbf{p}(\mathbf{x})$*)* function to produce the population of the next iteration.

**Algorithm 4** Estimation of Distribution Algorithms (EDAs)

```
P ← GenerateInitialPopulation()
while termination conditions not met do
  P_sel ← ChooseFrom(P) {{P_sel ⊆ P}}
  p(x) = p(x|P_sel) ← EstimateProbabilityDistribution(P_sel)
  P ← SampleProbabilityDistribution(p(x))
end while
```

The field of EDAs is still quite young and much of research effort is focused on methodology rather than high-performance applications. Applications of EDAs to the knapsack problem, the job shop scheduling (JSS) problem, and other CO problems can be found in [152].

#### 2.2.1.4 Ant Colony Optimization - ACO

*Ant Colony Optimization* is a metaheuristic in which a colony of artificial ants cooperate in finding good solutions to difficult discrete optimization problems[6] [39, 61, 65, 69, 106]. A part of study of this thesis is about ACO thus

[6]Most of works in literature have focused on ACO's applications in discrete domain. Very recently, the question about ACO's applicability into continuous domain has been raised and early works have been carried out. More

an extensive literature review about this metaheuristic is presented in Chapter 3. To make the report consistent, in this section we will show a general overview about ACO.

The first algorithm belonging to the framework of ACO metaheuristic was Ant System ( [67]). Poor performance of AS is improved by a number of its different algorithmic variants, for instances, Ant Colony System (ACS) ( [65]), Max-Min Ant System (MMAS) ( [190, 191]), and Best-Worst Ant System (BWAS) ( [43, 44]. These algorithmic variants come under a common framework as a consequence of research efforts first carried out in [63, 64]. The term *Ant Colony Optimization* has appeared since then.

Cooperation is the key design component of ACO algorithms: The choice is to allocate the computational resources to a set of relatively simple agents (artificial ants) that communicate indirectly by *stigmergy*, that is, by indirect communication mediated by the environment.

ACO algorithms can be used to solve both static and dynamic combinatorial optimization problems[7].

In ACO algorithms, a number of *artificial ants* are used to search for solu-

reviews about this trend will be presented in Chapter 3.

[7]Static problems are those in which the characteristics of the problem are given once and for all when the problem is defined, and do not change while the problem is being solved. A paradigmatic example of such problems is the Traveling Salesman Problem (TSP) ( [131]) in which city locations and their relative distances are part of the problem definition and do not change at run time. On the contrary, dynamic problems are defined as a function of some quantities whose value is set by the dynamics of an underlying system. When the problem instance's characteristics dynamically change at running

tions. Each artificial ant is a stochastic procedure that constructs a new solution "from scratch" by adding opportunely defined solution components to an initially empty partial solution. Components adding is done until a solution is complete or other stopping criteria are met. During the solution construction stage, artificial ants use a shared global memory - that stimulates the stigmergy characteristic of real ants when finding the shortest tours. After ants finished building their solutions, the global memory is updated by some ants. Ants allowed to update the global memory are chosen in terms of some heuristic rules. The variance of the global memory before and after being updated is a function of the global memory in the past and quality of solutions constructed by chosen ants.

Many studies in the field found in literature suggest that performance of ACO algorithms is much improved if local search procedures are incorporated into these algorithms. Most implementations of ACO algorithms use a local search procedure at the time before the global memory is updated.

Successful applications of ACO include those in communication networks routing [26], the sequential ordering problem (SOP) [82], and resource constraint project scheduling (RCPS) [137]. Further references to applications of ACO can be found in [69]. Additionally refer to [62] for the most recent survey about ACO metaheuristic.

time the algorithm for the dynamic COPs must be capable of adapting online to the changing environment. An example of this situation is network routing problems in which the data traffic and the network topology can vary in time.

### 2.2.2 Trajectory Approaches

In this subsection, a number of state-of-the-art trajectory (or single-point-based) metaheuristics will be presented. They are consisting of Greedy Randomized Adaptive Search Procedure, Variable Neighborhood Search, Simulated Annealing, and Tabu Search.

#### 2.2.2.1 Local Search Methods

*Local search* is a general approach for finding high-quality solutions to hard combinatorial optimization problems in reasonable time. It is based on what is perhaps the oldest optimization method - *trial and error*. The idea is so simple and natural, in fact, that it is surprising how successful local search has proven on a variety of difficult combinatorial optimization problems.

Local search is basically based on the iterative exploration of *neighborhoods* of solutions trying to improve the current solution by *local changes*. The types of local changes that may be applied to a solution are defined by a neighborhood structure. Given an optimization problem whose set of feasible points is $\mathcal{S}$, a neighborhood structure is define as following.

**Definition 2.2.1** (neighborhood structure)**.** *A neighborhood structure is a function* $\mathcal{N} : \mathcal{S} \mapsto 2^S$ *that assigns a set of neighbors* $\mathcal{N}(s) \subseteq \mathcal{S}$ *to every* $s \in \mathcal{S}$. $\mathcal{N}(s)$ *is also called the neighborhood of* $s$.

If $\mathcal{S} = \mathcal{R}^n$, the set of points within a fixed Euclidean distance obviously provides a natural neighborhood. The choice of an appropriate neighborhood structure is crucial for the performance of a local search algorithm and is problem-specific. The neighborhood structure defines the set of solutions

that can be reached from $s$ in *one single step* of a local search algorithm. Typically, a neighborhood structure is defined implicitly by defining the possible local changes that may be applied to a solution, and not by explicitly enumerating the set of all possible neighbors.

The solution found by a local search algorithm may only be guaranteed to be optimal with respect to local changes and, in general, will not be a globally optimal solution.

For example, the well-known *k-opt* (or also called *k-change*) neighborhood for traveling salesman problem (TSP) ( [160]) is one of such implicitly defined neighborhood structures. Given a tour $f$ of TSP, *k-opt* is defined as follows:

**Definition 2.2.2** (k-opt)**.**

$$\mathcal{N}_k(f) = \{g : g \in \mathcal{S} \textit{and } g \textit{ can be obtained from } f \textit{ as follows: remove } k \textit{ edges from the tour } f\textit{; then replace them with } k \textit{ edges}\}$$

**Definition 2.2.3** (local optimal)**.** *Given an optimization problem with a neighborhood structure $\mathcal{N}$ and a cost function $f$, a local optimal with respect to $\mathcal{N}$ or simply local optimal is a solution $s$ such that $\forall s' \in \mathcal{N}(s) : f(s) \leq f(s')$.*

A local search algorithm also requires the definition of a neighborhood examination scheme that determines how the neighborhood is searched and which neighbor solutions are accepted. While the neighborhood can be searched in many different ways, in the great majority of cases the acceptance rule is either the *best-improvement* rule, which chooses the neighborhood solution giving the largest improvement on the objective function, or the *first-improvement* rule, which accepts the first improved solution found.

In its most basic version, often called *iterative improvement*, or sometimes *hill-climbing* or *gradient-descent* for maximization or minimization problems,

respectively, the local search algorithm, see Algorithm (5), searches for an improved solution within the neighborhood of the current solution. If an improved solution is found, it replaces the current solution and the local search is continued. These steps are repeated until no improving solution if found in the neighborhood and the algorithm terminates in a local optimal. A disadvantage of iterative improvement is that the algorithm may stop at a very poor-quality local optimal. Another popular local search for the solution of nonlinear non-convex optimization problems is the *trust regions* or *restricted step* method [25, 28, 120] which approximates only a certain region (the so-called trust region) of the objective function with a quadratic as opposed to the entire function (for a comprehensive reference of this method, refer to [40]). When an adequate approximation of the objective function is found within the trust region then the region is expanded.

**Algorithm 5** The basic local search algorithm - iterative improvement.

**Sub-Procedure** improve(t)

$$\text{improve(t)} = \begin{cases} \text{any } s \in \mathcal{N}(t) \text{ with } f(s) < f(t) \text{ if such an } s \text{ exists} \\ \text{“no” otherwise} \end{cases}$$

**End Sub-Procedure**
t ← some initial starting point in $\mathcal{S}$;
**while** improve(t) ≠ ‘no’ **do**
  t ← improve(t)
**end while**
**return** t

To apply this basic local search to a particular problem, we must decide a number of choices. First, we must decide how to obtain an initial feasible solution. Next, we must choose a “good” neighborhood structure for the problem at hand and an effective and efficient method for searching it. This choice is usually guided by intuition, because very little theory is available

as a guide [37, 129, 160]. And the design of effective local search algorithms has been, and remains, very much an art.

**2.2.2.1.1 Greedy Randomized Adaptive Search Procedure - GRASP** The metaheuristic *greedy randomized adaptive search procedures* also known as GRASP [73, 175] is a multi-start or iterative process. It randomizes greedy construction heuristics to allow the generation of a large number of different starting solutions for applying a local search. In the iterative process, each iteration basically consists of two stages: construction and local search. Assume that the set of candidate components is formed by all components which can be incorporated into the partial solution under construction without violating feasibility. The algorithmic framework in Alg. (6) shows a high-level pseudo code of GRASP.

**Algorithm 6** Greedy randomized adaptive search procedures - GRASP

```
for i = 1, ..., Number-of-Iterations do
  s ← GreedyRandomizedConstruction(seed) { seed is used for a pseudo-
  randome number generator}
  s ← LocalSearch(s)
  UpdateSolution(s, best-solution)
end for
return best-solution
```

In the former stage a feasible solution is built from scratch by a constructive heuristic that is based on a greedy function. In this construction process, a next component is determined according to the evaluation of a greedy function for all candidate components. This greedy function represents the gradual increase of cost function because of the incorporation of a new component into the current partial solution under consideration. The component is randomly selected from a *restricted candidate list* (RCL) whose elements are

formed by best elements, i.e. those resulting in the smallest incremental costs when incorporated into the current partial solution[8]. Typical ways of deploying the restricted candidate list are either to take the best $\theta\%$ of the solution components or to take all solution components whose value of the greedy evaluation function is within $\gamma\%$ of the best-evaluated solution component.

After having a component incorporated, the RCL is updated and the incremental cost is reevaluated[9]. This strategy is similar to the so-called *semi-greedy heuristic* proposed in [114]. The solutions obtained by a greedy randomized construction is not highly possible not an optimal. The local stage usually enhances the constructed solutions.

A local search algorithm operates in an iterative way by successively replacing the current solution by a better solution in the neighborhood of the current solution (a very similar behavior to this iterative fashion can be found in Algorithm (5) on page 53). However, also note that the basic GRASP described above indicates that it does not rely on the history of the search process. Only memory for a problem instance and best-so-far solution is required.

GRASP can be efficient if at least two following conditions are met: 1) the construction heuristic is able to sample the most promising areas of the search space; 2) the solutions constructed belong to the basins of attraction of different local minima.

Numerous available applications of GRASP and several variants of the basic GRASP algorithm presented above have been proposed, for an exten-

[8]This is the greedy aspect of the metaheuristic hence its name.

[9]This is the adaptive aspect of the metaheuristic.

sive survey see [74]. Among of those applications are feedback vertex set problem [162], quadratic assignment problem [161, 175], and maximum clique problem [119]. It is noteworthy that, theoretically, using RCL in the standard implementation of GRASP may negatively affect on the convergence of the heuristic to the optimal solutions ( [145] for the regarded theoretical result). One way to go around this problem is to let the parameter $\theta$ be randomly chosen according to a uniform distribution [173].

#### 2.2.2.1.2 Variable Neighborhood Search - VNS

*Variable Neighborhood Search* is a recently proposed metaheuristic for solving combinatorial and global optimization problems [112, 113, 144]. Its idea is derived from a simple principle: change the neighborhood structure within a local search when the search is trapped on a local optima.

**Algorithm 7** Basic scheme of variable neighborhood search - VNS

*Initialization*. Select the set of neighborhood structures $\mathcal{N}_k$ for $k = 1, \ldots, k_{max}$, that will be deployed in the search; find an initial solution $x$;
**repeat**
  Set $k \leftarrow 1$;
  **while** $k < k_{max}$ **do**
    **(a) Shaking**. Generating a point $x'$ at random from the $k^{th}$ neighborhood of $x$ ($x' \in \mathcal{N}_k(x)$);
    **(b) Local search**. Given $x'$ obtained from (a), a local search is applied with $x'$ as the initial solution. Denote $x''$ as the local optimum solution obtained from this local search;
    **(c) Jump or not**. If $x''$ is better than the incumbent solution, replace $x$ by $x''$ and start a completely new search progress with the neighborhood structure $\mathcal{N}_{k \leftarrow 1}$[10]. Otherwise, increase $k$ by 1.
  **end while**
**until** *termination conditions are met*
**return** best-solution

The simplicity of the basic scheme of VNS gives many degrees of freedom for designing variants and particular instantiations. VNS is based on the

three following facts [144]:

- ☞ **Fact 1** A local minimum w.r.t. one neighborhood structure is not necessary so with another.
- ☞ **Fact 2** A global minimum is a local minimum w.r.t. all possible neighborhood structures.
- ☞ **Fact 3** For many problems local minima w.r.t. one or several neighborhood structures are very relatively close to each other.

There are three different ways to use facts 1 and 3 to solve an optimization including i) deterministic; ii) stochastic; iii) both deterministic and stochastic. The Algorithm (7) describes the basic scheme of VNS that combines both deterministic and stochastic changes of neighborhood structures.

For the sake of definitiveness, in the following subparagraphs we present the schemes of the deterministic way and the stochastic one whose technical names in literature are **Variable Neighborhood Descent** (VND) and **Reduced VNS** (RVNS) respectively.

**Variable Neighborhood Descent - VRD** is a basic variant of VNS when the change of neighborhood structure is done in a deterministic way. VND's algorithmic steps are presented in Algorithm (8).

Usually, local search heuristics use in their descent a single or rarely two neighborhoods, or in other words $k_{max} \leq 2$. Also notice that the final solution will be a local optimum with respect to all $k_{max}$ neighborhood structures, so higher chance to reach a global optimum solution is given to the strategy of using more than a single neighborhood structure. However, since this VNS

**Algorithm 8** Variable Neighborhood Descent - VND

*Initialization*. Select the set of neighborhood structures $\mathcal{N}_k$ for $k = 1, \ldots, k_{max}$, that will be deployed in the decent; find an initial solution $x$ (or apply a certain rule to a given $x$);
**repeat**
  Set $k \leftarrow 1$;
  **while** $k < k_{max}$ **do**
    **(a) Neighborhood exploring**. Find the best neighbor $x'$ of $x$ according to the $k^{th}$ neighborhood structure $\mathcal{N}_k$ by a certain local search;{Similar to Alg. (5) on page 53}
    **(b) Jump or not**. If the found solution $x'$ is better than the best-so-far solution $x$ then $x \leftarrow x'$, $k \leftarrow 1$, otherwise set $k \leftarrow k + 1$,
  **end while**
**until** *no improvement is obtained*
**return** best-solution

variant will search the whole neighborhood of a given solution for the best local optimum of that neighborhood, much more computational resource must be dedicated to it when the size of problem instance increases. To overcome this difficulty, an alterative way is to use the stochastic approach named Reduced VNS that is presented in the following subparagraph.

**Reduced VNS - RVNS** is efficient and useful when dealing with large problem instances for that VND is computationally expensive. Empirical studies suggest that the best value for $k_{max}$ should be 2 [144]. Moreover, one of termination conditions utilizes the maximum number of iterations between two improvements.

Among applications of VNS are vehicle routing problem [23], maximum clique problem [111], and minimum spanning tree problem [174].

**Algorithm 9** Reduced VNS - RVNS

*Initialization*. Select the set of neighborhood structures $\mathcal{N}_k$ for $k = 1, \ldots, k_{max}$, that will be deployed in the search; find an initial solution $x$;
**repeat**
  Set $k \leftarrow 1$;
  **while** $k < k_{max}$ **do**
    **(b) Shaking**. Produce in random a neighbor $x'$ from the $k^{th}$ neighborhood $\mathcal{N}_k$ of $x$.
    **(b) Jump or not**. If the produced solution $x'$ is better than the current solution $x$ then $x \leftarrow x'$, $k \leftarrow 1$; in other words the search is going on with the neighborhood structure $\mathcal{N}_{k \leftarrow 1}$, otherwise set $k \leftarrow k + 1$,
  **end while**
**until** *termination conditions are met*
**return** best-solution

#### 2.2.2.2 Simulated Annealing - SA

*Simulated Annealing* is commonly considered as the oldest among the metaheuristics. The origins of the algorithm are in statistical mechanics (see the Metropolis algorithm [138, 139]). The idea of SA was provided by the annealing process of meta and glass, which assume a low energy configuration when cooled with an appropriate cooling schedule. Among recent trends of extending SA, using *quantum fluctuations* [51] instead of thermal fluctuations is to get through high but thin barriers in the target function or the idea of *stochastic tunneling* [109] which attempts to overcome the increasing difficulty simulated annealing runs have in escaping from local minima as the temperature decreases, by 'tunneling' through barriers.

SA was first presented as a search algorithm for CO problems in [123] and [200]. In order to avoid getting trapped in local minima, the fundamental idea is to allow moves to solutions with objective function values that are worse than the objective function value of the current solution. Such a kind of move is often called an hill-climbing move. At each iteration, a solution

$s' \in \mathcal{N}(s)$ is randomly chosen. If $s'$ is better than $s$ (i.e. has a lower objective function value), then $s'$ is accepted with a probability which is a function of a temperature parameter $T_k$ and $f(s') - f(s)$. This probability is computed following the Boltzmann distribution:

$$\mathbf{p}\,(s'|T_k, s) = e^{-\frac{f(s') - f(s)}{T_k}} \tag{2.2.1}$$

**Algorithm 10** Simulated Annealing (SA)

```
s ← GenerateInitialSolution()
k ← 0
T_k ← SetInitialTemperature()
while termination conditions not met do
  s' ← PickNeighborAtRandom(N(s))
  if f(s') < f(s) then
    s ← s' {s replaces s}
  else
    Accept s' as new solution with probability p(s'|T_k, s) {{ see ( 2.2.1 ) }}
  end if
  AdaptTemprature(T_k)
end while
```

- AdapTemperature($T_k$): The temperature $T_k$ is adapted at each iteration according to a *cooling schedule* (or cooling scheme)[11].

SA has been applied to many CO problems, such as the quadratic assignment problem (QAP) [41] and job shop scheduling (JSS) problem [127]. References

[11]The cooling schedule is a critical component to SA to determine the cooling rate to be low enough for the probability distribution of the current state to be near the equilibrium at all times. In practice, the ideal cooling rate is not able to be determined in advance which makes designing a cooling schedule a difficult task. There are many approaches to adapt $T_k$ found in literature like [177, 179]

to other applications can be found in [75, 118]. SA is nowadays more used as a component in metaheuristics, rather than applied as a stand-alone search algorithm.

#### 2.2.2.3 Tabu Search - TS

*Tabu search* is one of the most successful metaheuristics for the application to CO problems [20]. The basic ideas of TS were introduced in [89], based on earlier ideas formulated in [88]. A description of the method and its concepts can be found in [96]. The basic idea of TS is the explicit use of search history, both to escape from local minima and to implement an explorative strategy. A simple version of TS is described in this section in order to introduce the basic concepts. A simple TS algorithm (see Algorithm (11)) is based on a best-improvement local search and uses a *short term memory* to escape from local minima and to avoid a cycle[12]. Recent application trends on TS research is to incorporate TS with other search techniques (like EC) [158] or with mathematical programming methods (like Simplex) [164].

**Algorithm 11** Tabu Search (TS)

```
s ← GenerateInitialSolution()
TL ← ∅
while termination conditions not met do
  N_a(s) ← N(s)\TL
  s' ← arg min {f(s'')|s'' ∈ N_a(s)}
  Update(TL, s, s')
  s ← s' {s replaces s'}
end while
```

The short term memory is implemented as a *tabu list TL* that keeps track of the most recently visited solutions and excludes them from the neighbor-

[12] A cycle is a sequence of moves that constantly repeats itself.

hood of the current solution. The restricted neighborhood of a solution $s$ is referred as the *allowedset*, which is denoted by $\mathcal{N}_a(s)$. At each iteration, the best solution from the allowed set is chosen as the new current solution. In addition, in procedure $Update(TL, s, s')$ this solution is added to the tabu list and - if $TL$ has reached its maximum capacity - one of the solutions that were already in the tabu list is removed.

TS has been applied to most CO problems; examples of successful applications are the Robust Tabu Search to the QAP [194], the Human-Guided Tabu Search to the 2D HP protein folding problem (HPPF) [133], and to assignment problems [53]. TS approaches dominate to job shop scheduling (JSS) problem area ( [153] and vehicle routing problem (VRP) area [85]. Further references of applications can be found in [96].

## 2.3 Improving Performance of Metaheuristics

### 2.3.1 Hybridization

For many years, the research of metaheuristics has mainly focused on the application of single metaheuristics to plenty of hard problems in variety of areas like engineering, logistics, bioinformatics, etc. Recently, it has become clear that applicability of a sole metaheuristic is rather limited when the dimensional complex of underlying problems increases. A skillful combination of concepts of different metaheuristics, a so-called hybrid metaheuristic, can provide a more efficient behavior and higher flexibility when dealing with real-world and large-scale problems.

#### 2.3.1.1 Memetic Approaches

*Memetic Algorithms* (MAs) is a population-based approach for heuristic search in optimization problems. The term memetic algorithms was firstly used in [146] but originated from the term *meme* from a well-known book on the evolution theory [52][13]. The evolutionary difference between "memes" and "genes" is that memes are possibly improved and processed *locally* meaning that the people hold them can process and possibly improve them - something that cannot take place to genes. Locality is the key to that difference and that is also the idea MAs are based on.

A key feature, presented in most MAs implementations, is the use of a population-based search which intends to use all available knowledge about the problem similarly to the feature of Evolutionary Algorithms (EAs). The difference between MAs and EAs is that in MAs:

> This knowledge is incorporated in the form of heuristics, approximation algorithms, local search techniques, specialized recombination operators, truncated exact methods and many other ways

[13]The following is R. Dawkins's own words about memes: "Examples of memes are tunes, ideas, catch-phrases, clothes fashions, ways of making pots or building arches. Just as genes propagate themselves in the gene pool by leaping from body to body via sperms and eggs, so memes propagate themselves in the meme pool by leaping from brain to brain via a process which, in the broad sense, can be called imitation." An alternative name of MAs is *Cultural Evolution* due to this. Other names of MAs include Hybrid Genetic Algorithms, Genetic Local Search, Lamarckian Genetic Algorithms, Baldwinian Genetic Algorithms.

(chapter 14 in [147]).

While Genetic Algorithms (GAs) ( [100, 140, 169]) was inspired by stimulating biological evolution, MAs is to mimic cultural evolution (see footnote 13)[14]. In MAs, each individual (agent) of population is allowed to perform a heuristic local search to enhance its fitness until reaching a pre-determined level. When this level of fitness is reached, the individual will start its either *cooperative* or *competitive* interaction with other members in the population. The competition can be similar to the selection process of GAs, while the cooperation can be any "breeding" methods that result in a new individual like the mechanism of crossover in GAs. In general, cooperation can be considered as the interchange of information.

In essence, most MAs can be interpreted as a *cooperative-competitive* strategy of optimizing agents [146]. The central idea of this strategy, when the search space is large, is to use collective properties of a group of recognized agents which are separately performing the search. In consequence, the collective behavior of the whole population generates solutions that are better than those generated by each agent without interaction within the group.

A general framework of Local-Search-based Memetic Algorithms (LA-based MAs) is described in Alg. (12). LA-based MAs have been generally applied as a heuristic to find near-optimal solutions to problems that are proven to be *NP*-hard. By fixing any parameters given in any LA-based MAs, we must consider these LA-based MAs as a heuristic, or in other words an

[14]Based on this characteristic, we classify MAs into class of hybridization metaheuristics. Although MAs can naturally be grouped into Population-based Metaheuristics.

instance of a general metaheuristic, for the problem under consideration.

**Algorithm 12** A general framework of Local-Search-Based Memetic Algorithms - LA-based MAs

```
Initialize the population Pop[15]
for all individual i ∈ Pop do
  i ← Improve-and-Evaluate(i)
end for
repeat
  Pop ← LS-Recombine(Pop) {Refer to Alg. (13) for details of this subroutine.}
  Pop ← LS-Mutate(Pop) {Refer to Alg. (13) for details of this subroutine.}
  Pop ← Select(Pop)
  if no improvements for Pop then
    Pop ← Restart(Pop)
  end if
until termination-conditions-satisfied
return best solutions found
```

The general framework of a LS-based MAs in Alg. (12) is mostly the same as the framework of a GAs. However, their difference lie in the deeper levels. Before an offspring created by the same way as done in the procedure **Recombine** of any GAs, the subroutine **LS-Recombine** within the loop of $N_{rec}$ steps of recombination is evaluated and inserted into the population. Thus the offspring's fitness is improved by a certain heuristic local search-based procedure. This procedure will try to iteratively find a better individual until it is impossible to do so. The function of the procedure **LS-Mutate** is also explained by the same way. Moreover, the usage of the procedure **Improve-and-Evaluate** in the procedure **LS-Mutate** means that individuals will have a possibility of being re-optimized after they passed the recombination process. The purpose of using the procedure **Restart** is to try to avoid a premature convergence of the search process.

MAs has successfully applied to multitude of real-world problems like

**Algorithm 13** Subroutines LS-Recombine() and LS-Mutate() in LA-based MAs

**Sub-Function** *Improve-and-Evaluate*(individual $i_m$)

individual $i_m \leftarrow$ Local-Search-based-Improver(individual $i_m$)
individual $i_m \leftarrow$ Evaluate(individual $i_m$)

**End Sub-Function**
**Sub-Routine** *LS-Recombine(Pop)*
**for** $n = 1, \ldots, N_{rec}$ **do**
*offspring* $i_n \leftarrow$ Recombine(*Pop*) {This routine operates equivalently to that in EC, see subsection 2.2.1.1;}
*offspring* $i_n \leftarrow$ **Improve-and-Evaluate**(*offspring* $i_n$)
add *offspring* $i_n$ into *Pop*
**end for**
**End Sub-Routine**
**Sub-Routine** *LS-Mutate(Pop)*
**for** $m = 1, \ldots, M_{mut}$ **do**
*individual* $i_m \leftarrow$ Mutate(*Pop*) {This routine operates equivalently to that in EC, see subsection 2.2.1.1;}
*individual* $i_m \leftarrow$ **Improve-and-Evaluate**(*individual* $i_m$)
add *individual* $i_m$ into *Pop*
**end for**
**End Sub-Routine**

Machine Scheduling Problem [157], Timetabling [24], Protein Structure Prediction Problem [126], Vehicle Routing Problem [16]. A rather detailed reference about applications and implementations of is given in a MAs tutorial in [125] or in the survey [181].

### 2.3.2 Exploiting Problem Structure

Improving performance of metaheuristic algorithms by embedding local search procedures into them is widely confirmed by extensively empirical works. However, there are some issues related to the usage of the local search procedures for which we need to carefully pay attention at the algorithm design stage. One issue is of how to design a "good" neighborhood structure. Another issue is that given a neighborhood structure, how to tune systematic parameters of the algorithm and where and when to invoke these procedures during the runtime. This section delivers a discussion about the former issue only. Given an optimization problem, practically a neighborhood structure is regarded to as a "good" one if it is specifically designed for that problem, or in other words it must reflect, as many as possible, specific and useful structures of that problem. An successful example of such neighborhood structures is a so-called *k-change* neighborhood [134] that is popularly used as a local search in several algorithms applied to Traveling Salesman Problem (see subsection 3.3.1.1 on page 106 for a definition of Traveling Salesman Problem).

The next subsection will take a look at the landscape theory built on Grover's difference equation. This theory has been used to search for favorable properties to local search. Subsection 2.3.2.2 presents a review of recent works

constructing local structures (of combinatorial optimization problems) whose characteristics and performance are analyzed and evaluated in terms of group theory - a powerful tool has been applied to study exact search methods.

#### 2.3.2.1 Find Useful Search Neighborhoods using Landscape Theory

Recently, a great number of theoretical works that focus on searching properties favorable to local search has been proposed and developed. Among of them is the *landscapes theory*, seemingly, beginning with the work in [105] and in [31] and being extended by [11, 184, 185]. An abstract definition of landscape is given in the following definition.

**Definition 2.3.1** (Landscape)**.** *Given a combinatorial optimization problem whose solution space is $\mathcal{X}$, a neighborhood structure $\mathcal{N}$, and an objective function $f : \mathcal{X} \mapsto \mathbb{R}$, the triple $\mathcal{L} = (\mathcal{X}, f, \mathbb{R})$ defines a landscape of that problem with respect to the neighborhood structure $\mathcal{N}$.*

The Grover's difference equation [105] (that is similar to the wave function of mathematical physics) states that if a landscape satisfies it then any local optimal is **superior** to the average value $\mu$ of the objective function over the solution space[16], and, the number of steps to reach a solution at least as good as $\mu$ from **any starting point is linear in the problem size**. Landscapes with specific notions of neighborhood arise from certain classes of combinatorial optimization problems, for example symmetric traveling salesman problem, min cut problem, and graph partitioning problem were proved to satisfy Grover's difference equation [31, 105].

[16]This means that arbitrarily poor local optima do not exist.

**Grover's Difference Equation** Before introducing the equation, we need to present some preliminary notions need. Let $\mathcal{C}$ be the cost function of a combinatorial optimization problem[17], $\mathcal{C}_{avr}$ be the average value over all the costs of all feasible solutions to this problem. With the definition $f(x) = \mathcal{C}(x) - \mathcal{C}_{avr}$, the Grover's difference equation is defined as

$$\nabla^2 f + \frac{k}{n} f = 0, \quad k > 0, \tag{2.3.1}$$

where $n$ is the problem size, and $k$ is a constant depending on the problem. The symbol $\nabla^2 f$ is used to denote the average difference operator over a specific neighborhood. If a neighborhood structure[18] $\mathcal{N}$ is defined, then

$$\nabla^2 f(x) = \frac{\sum_{i=1}^{|\mathcal{N}(x)|} \delta_i}{|\mathcal{N}(x)|},$$

where $\delta_i$ is defined as

$$\delta_i = f(x) - f(x_i), \quad \forall x_i \in \mathcal{N}(x).$$

The difference equation (2.3.1) gives a relation between the (cost) $f(x)$ of a solution $x$ and $\nabla^2 f(x)$ (i.e. the costs of the solutions belonging to the neighborhood of $x$). This "link" between $f(x)$ and $\nabla^2 f(x)$ gives some information about the local structure of the problem that satisfy (2.3.1. This information is *local* because at each step of the local search procedure, we have some relation between the solution $x$ and its neighbors, however, no relation can be established between $x$ and the previous or the next solutions examined in the local search procedure.

[17] $\mathcal{C}(x)$ denotes the cost of solution $x$.

[18] The original difference equation in [105] considered neighborhood structures that are regular and symmetric.

#### 2.3.2.2 Construct and Characterize Search Neighborhoods using Group Theory

There are two important factors in designing a local search procedure that are separately considered. One is the definition of the neighborhood structure. The other is the "instructions" of how to move that structure from one (or a set of) point(s) to another (or other set of points) *efficiently*. When the maximum cardinal (size) of set of neighbors of any point in the solution space is so large, using an exhaustive search procedure will be costly. Although depending on characteristics of the neighborhood structure themselves, it is more possible that maximum cardinal increases when size of instances of problems under consideration increases.

Thus, given a neighborhood structure, efficiently moves will positively affect the outcome of the local search procedure. To heuristics and metaheuristics, group theory has emerged as one of promising approaches ( [12, 34, 36, 46] not only to study how to find such moves by, for example preserving some "good" sub-paths (or predetermined sub-structures) in elite solutions, but also to find out other types of neighborhood structures. Historically, group theory is the foundation of several exact methods of integer and mixed-integer programming ( [87, 101, 102, 207]) but although being a promising approach, group theory has only recently been in limited use in heuristic and metaheuristic methods [12, 34–36, 38, 46, 122].

The following instantiation gives a simple picture of how group theory can be applied to construct and characterize search neighborhoods by considering a special case of *k-opt* neighborhood structure for the TSP used as an illustrative instance.

With reference to the very illustrative definition of *k-opt* local search in the definition (2.2.2) (on page 52), the neighborhood structure of *k-opt* will be explicitly defined by using *basic moves* or *single steps* concepts that are, in turn, based on the concept of *permutation cycle* in group theory. Studying the factor groups of the non-Abelian symmetric group $S_n$ on $n$ symbols can reveal neighborhood structures of the incumbent solution of a $n$-city TSP instance [19]. Indeed, to give a simple example, let us consider a simple case when $k = 2$ (the well-known 2−opt local search applied to TSP).

Let $T_n$ be the set of all *transpositions* ( or permutation cycles of length 2) of the symmetric group $S_n$. Definition of a transposition is in ( 2.3.2 )

$$T_n = \{a_{ij} = (i, j) : \ \forall 1 \leq i < j \leq n\}. \tag{2.3.2}$$

Given a permutation $\tau = (\tau_1 \ \tau_2 \dots \tau_n)$, when applying a transposition, says $(i, j)$ to $\tau$, we obtain a new permutation $\pi = (\pi_1 \ \pi_2 \dots \pi_n)$ from $\tau$ by swapping positions of the $i^{th}$ and $j^{th}$ objects in $\tau$.

Thus, it is clear to see that, $T_n$ is the complete set of basic steps of the neighborhood structure *2-opt*. Thus, the *2-opt* neighborhood is algebraically described.

To algebraically describe the other cases when $k > 2$, for example $k = 6$, and let $U \equiv$ the union of *conjugacy classes* whose cycle structures are $(x, x, x, x, x, x)$, $(x, x)(x, x, x, x)$ and $(x, x, x)(x, x, x)$, then the neighborhood of an element $p \in S_n$ is given by $p^C$.

Recently found results [35, 36] substantiate the relevance of applying group theory to the study of metaheuristic neighborhoods. A consequence of those

[19] Remember that since a solution of a $n$-city TSP instance can be represented as a permutation of $\{1, 2, \dots, n\}$.

results is the counter-intuitive fact for the presumption on which many search strategies are based. The presumption is that an incumbent tour's length will directly reflect its neighborhood weight (the neighbors's summed tourlengths), i.e. a good tour will be found in a neighborhood of good tours. The counter-intuitive fact found by group theoretic analysis in [36] shows that, under certain neighborhood types, a very inferior solution (associated with a relatively large length) could have a small neighborhood weight implying that there exist short tours in the neighborhood of this inferior solution. Findings in [36] also gave an theoretical evidence of the success seen in the application of the swap neighborhood in literature (for example, see [132].

Moreover, so-called *conjugative rearrangement* neighborhoods that preserve the *cycle structure* of the incumbent solution, i.e. all candidate neighbor solutions must have the same number of cycles and each cycle must have the same number of cities as in the incumbent solution, was proposed in [35, 36]. Considerations of neighborhoods that do not preserve the cycle structure of the incumbent solution can be found in [37].

## 2.4 Summary of Chapter

In this chapter, we have presented and compared most important metaheuristic methods nowadays. In Subsections 2.2.1 and 2.2.2 we have outlined the basic configurations of metaheuristics as they are represented in literature. Approaches to improving performance of metaheuristics have been presented in this chapter as well. Integration of classical methods in AI and OR into metaheuristics for dealing with larger instances of optimization problems is described in Section 2.3. Actually, a part of our works focused on integrating

clustering methods into search metaheuristic to reduce running time will be presented in Section 4.2. Moreover, based on the well-established group theory, a novel active and promising approach to gain systematically insights of factors affecting on performance of local search procedures has been revised in this chapter.

# Chapter 3

# Ant Colony Optimization

Among state-of-the-art metaheuristics, Ant Colony Optimization (ACO) has received attention of many researchers.

In this chapter, basic principles of ACO and its recent developments is deserved in section 3.1. Then, in section 3.2 our proposal and convergent analysis for a generalized variant of Graph-based Ant System [106] are presented. Results of this work was published and can be accessed at [59]. Following findings of the convergent analysis in section 3.2, a study on how to improve performance of Ant-based algorithms by dynamically tuning their systematic parameters, specifically by tuning the exploiting parameter, is denoted in section 3.3. The empirical findings in that study was published and can be found at [58]. The last section will give a summary of this chapter.

## 3.1 Background

Ant colony optimization (ACO) [61, 63, 64, 66, 69] is a metaheuristic approach that was inspired by the foraging behavior of real ants. This behavior - as described by Deneubourg et al. in [54] - enables ants to find shortest paths between food sources and their nest. Initially, ants explore the area surrounding their nest in a random manner. As soon as an ant finds a source of food, it evaluates quantity and quality of the food and carries some of this food to the nest. During the return trip, the ant deposits a chemical pheromone trail on the ground. The quantity of pheromone deposited, which may depend on the quantity and quality of the food, will guide other ants to the food source.

The indirect communication between the ants via the pheromone trails allows them to find shortest paths between their nest and food sources. This functionality of real ant colonies is exploited in artificial ant colonies in order to solve CO problems.

In ACO algorithms, the chemical pheromone trails are simulated via a parameterized probabilistic model that is called *pheromone model*. The pheromone model consists of a set of model parameters whose values are called the *pheromone values*. The basic ingredient of ACO algorithms is a constructive heuristic that is used for probabilistically constructing solutions using the pheromone values.In general, the ACO approach attempts to solve a CO problem by iterating the following two steps:

- Solutions are constructed using a pheromone model, that is, a parameterized probability distribution over the solution space.
- The constructed solutions and possibly solutions that were constructed in earlier iterations are used to modify the pheromone values in a way that is deemed to bias future sampling toward high quality solutions.

### 3.1.1 Problem Representation

An artificial ant in ACO is a stochastic constructive procedure that incrementally builds a solution by adding opportunely defined solution components to a partial solution under construction. Therefor, the ACO metaheuristic can be applied to any combinatorial optimization for which a constructive heuristic can be defined on.

Although this means that ACO metaheuristic can be applied to any inter-

esting combinatorial optimization problems, the real issue is how to map the considered problem to a representation that can be used by the artificial ants to build solutions. The following is a formal characterization of the representation that the artificial ants use and of the policy they implement.

Let us consider the minimization problem $(\mathcal{S}, f, \Omega)$, where $\mathcal{S}$ is the *set of candidate solutions*, $f$ is the *objective function* which assigns an objective function (cost) value $f(s,t)$ to each candidate solution $s \in \mathcal{S}$, and $\Omega(t)$ is a *set of constraints*. The parameter $t$ indicates that the objective function and the constraints can be time-dependent, as is the case, for example, in applications to dynamic problems.

The goal is to find a *globally optimal* feasible solution $s\star$,that is, a minimum cost feasible solution to the minimization problem.

The combinatorial optimization problem $(\mathcal{S}, f, \Omega)$ is mapped on a problem that can be characterized by the following list of items:

- A finite set $C = \{c_1, c_2, \ldots, c_{N_C}\}$ of *components* is given, where $N_C$ is the number of components.

- The *states* of the problem are defined in terms of sequences $x = \langle c_i, c_j, \ldots, c_h, \ldots \rangle$ of finite length over the elements of $C$. The set of all possible states is denoted by $\chi$. The length of a sequence $x$, that is, the number of components in the sequence, is expressed by $|x|$. The maximum length of a sequence is bounded by a positive constant $n < +\infty$.

- The set of (candidate) solutions $\mathcal{S}$ is a subset of $\chi$ (i.e. $\mathcal{S} \subseteq \chi$).

- A set of feasible states $\widetilde{\chi}$, with $\widetilde{\chi} \subseteq \chi$, defined via a problem-dependent test that verifies that it is not impossible to complete a sequence $x \in \widetilde{\chi}$

into a solution satisfying the constraints $\Omega$. Note that by this definition, the feasibility of a state $x \in \widetilde{\chi}$ should be interpreted in a *weak* sense. In fact, it does not guarantee that a completion $s$ of $x$ exists such that $s \in \widetilde{\chi}$.

- A non-empty set $\mathcal{S}^\star$ of optimal solutions, with $\mathcal{S}^\star \subseteq \widetilde{\chi}$ and $\mathcal{S}^\star \subseteq \mathcal{S}$.

- A *cost* $g(s,t)$ is associated with each candidate solution $s \in \mathcal{S}$. In most cases, $g(s,t) \equiv f(s,t),\ \forall s \in \widetilde{\mathcal{S}}$, where $\widetilde{\mathcal{S}} \subseteq \mathcal{S}$ is the set of feasible candidate solutions, obtained from $\mathcal{S}$ via the constraints $\Omega(t)$.

- In some cases a cost or the estimate of a cost, $J(x,t)$, can be associated with states other than candidate solutions. If $x_j$ can be obtained by adding solution components to a state $x_i$, then $J(x_i,t) \leq J(x_j,t)$. Note that $J(s,t) \equiv g(s,t)$.

Given this formulation, artificial ants build solutions by performing randomized walks on the completely connected graph $G_C = (C, L)$ whose nodes are the components $C$, and the set $L$ fully connects the components. The graph $G_C$ is called *construction graph*[1] and elements of $L$ are called *connections*
.

The problem constraints $\Omega(t)$ are implemented in the policy followed by the artificial ants, as explained in the next section. The choice of implementing the constraints in the construction policy of the artificial ants allows a certain degree of flexibility. In fact, depending on the combinatorial optimization problem considered, it may be more reasonable to implement the constraints in a hard way, allowing the ants to build only feasible solutions,

[1] another definition of concept construction graph can be found at the definition (3.2.1) on page 88.

or in a soft way, in which case the ants can build infeasible solutions that can be penalized as a function of their degree of infeasibility.

### 3.1.2 Behavior of Artificial Ants

In ACO algorithms, artificial ants are stochastic constructive procedures that build solutions by moving on the construction graph $G_C = (C, L)$, where the set $L$ fully connects the components $C$. The problem constraints $\Omega(t)$ are built into the ants' constructive heuristic. In most applications, ants construct feasible solutions. However, sometimes it may be necessary or beneficial to also let them construct infeasible solutions. Components $c_i \in C$ and connections $l_{ij} \in L$ can have associated a *pheromone trail* $\tau$ ($\tau_i$ if associated with components, $\tau_{ij}$ if associated with connections), and a *heuristic value* $\eta$ ($\eta_i$ and $\eta_{ij}$, respectively). The pheromone trail encodes a long-term memory about the entire ant search process, and is updated by the ants themselves. Differently, the heuristic value, often called *heuristic information*, represents a priori information about the problem instance or run-time information provided by a source different from ants. In many cases, $\eta$ is the cost, or an estimate of the cost, of adding the component or connection to the solution under construction. These values are used by the ants' heuristic rule to make probabilistic decisions on how to move on the graph.

Each ant $k$ of the colony has the following properties:

- It exploits the construction graph $G_C = (C, L)$ to search for optimal solutions $s^\star \in \mathcal{S}^\star$.
- It has a memory $\mathcal{M}^k$ that it can use to store information about the path

it followed so far. Memory can be used to 1) build feasible solutions; 2) compute the heuristic values; 3) evaluate the solution found; and 4) retrace the path backward.

- It has a *start state* $x_s^k$ and one or more *termination conditions* $e^k$. Usually, the start state is expressed either as an empty sequence or as a unit length sequence, that is, a single component sequence.

- When in state $x_r = \langle x_{r-1}, i \rangle$, if no termination condition is satisfied, it moves to a node $j$ in its neighborhood $\mathcal{N}^k(x_r)$, that is, to a state $\langle x_r, j \rangle \in \chi$. If one of the termination conditions $e^k$ is satisfied, then the ant stops. When an ant builds a candidate solution, moves to infeasible states are forbidden in most applications, either through the use of the ant's memory, or via appropriately defined heuristic values $\eta$

- It selects a move by applying a probabilistic decision rule. The probabilistic decision rule is a function of 1) the locally available pheromone trails ( or pheromone values) and heuristic values; 2) the ant's private memory storing its current states; and 3) the problem constraints.

- When adding a component $c_j$ to the current state, it can update the pheromone trail $\tau$ associated with it or with the corresponding connection.

- Once it has built a solution, it can retrace the same path backward and update the pheromone trails of the used components.

It is important to note that ants act concurrently and independently and that although each ant is complex enough to find a (probably poor) solution

to the problem under consideration, good-quality solutions can only emerge as the result of the collective interaction among ants. This is obtained via indirect communication mediated by the information ants read or write in the variables storing pheromone values.

### 3.1.3 ACO framework

An ACO algorithm can be imagined as the interplay of three procedures: SolutionsConstruction, PheromonesUpdate, and DaemonActions. The ACO metaheuristic framework consisting of these procedures is shown in Algorithm (14). These three algorithmic components that are gathered in the *ScheduleActivities* construct. The *ScheduleActivities* construct does not specify how these three activities are scheduled and synchronized. This is up to the algorithm designer.

**Algorithm 14** Ant Colony Optimization (ACO) Framework

```
while termination conditions not met do
  ScheduleActivities
      SolutionsConstruction()
      PheromonesUpdate()
      DaemonActions() {optional}
  ScheduleActivities
end while
```

**SolutionsConstruction():** As mentioned before in 3.1, the basic ingredient of ACO algorithms is a constructive heuristic for probabilistically constructing solutions. A constructive heuristic assembles solutions as sequences of solution components taken from a finite set of solution components $\mathfrak{C}$ =

$\{c_1, \ldots, c_n\}$. A solution construction starts with an empty partial solution $\mathfrak{s}^p = \emptyset$. Then at each construction step the current partial solution $\mathfrak{s}^p$ is extended by adding a feasible solution component from the set $\mathfrak{N}(\mathfrak{s}^p) \subseteq \mathfrak{C} \setminus \mathfrak{s}^p$, which is defined by the solution construction mechanism. The process of constructing solutions can be regarded as a walk (or a path) on the so-called *construction graph* $\mathfrak{G}_C = (\mathfrak{C}, \mathfrak{L})$ whose vertices are the solution components $\mathfrak{C}$ and the set $\mathfrak{L}$ are the connections. The allowed walks on $\mathfrak{G}_C$ are hereby implicitly defined by the solution construction mechanism that defines set $\mathfrak{N}(\mathfrak{s}^p)$ with respect to a partial solution $\mathfrak{s}^p$. The choice of a solution component from $\mathfrak{N}(\mathfrak{s}^p)$ is at each construction step done probabilistically with respect to the pheromone model, which consists of *pheromone trail parameters* $\mathfrak{T}_i$ that are associated to components $c_i \in \mathfrak{C}$. The set of all pheromone trail parameters is denoted by $\mathfrak{T}$. The values of these parameters - the *pheromone values* - are denoted by $\tau_i$. In most ACO algorithms, the probabilities for choosing the next solution component - also called the *state transition probabilities* or simply *transition probabilities* - are defined as follows:

$$\mathbf{p}(c_i|\mathfrak{s}^p) = \frac{\tau_i^\alpha \cdot \eta(c_i)^\beta}{\sum_{c_j \in \mathfrak{N}(\mathfrak{s}^p)} \tau_j^\alpha \cdot \eta(c_j)^\beta}, \forall c_i \in \mathfrak{N}(\mathfrak{s}^p), \tag{3.1.1}$$

where $\eta$ is a weighting function, which is a function that, sometimes depending on the current partial solution, assigns at each construction step a heuristic value $\eta(c_j)$ to each feasible solution component $c_j \in \mathfrak{N}(\mathfrak{s}^p)$. The values that are given by the weighting function are commonly called the *heuristic information*. Furthermore, $\alpha$ and $\beta$ are positive parameters whose values determine the relation between pheromone information and heuristic information. Most implementations of ACO set $\alpha = 2.0$ and $\beta = 1.0$.

**PheromoneUpdate():** In ACO algorithms, we can find different types of pheromone updates. First, we outline a pheromone update that is used by basically every ACO algorithm. This pheromone update consists of two parts. First, a *pheromone evaporation*, which uniformly decreases all the pheromone values, is performed. From the practical point of view, pheromone evaporation is needed to avoid a too rapid convergence of the algorithm toward a sub-optimal region. It implements a useful form of *forgetting*, favoring the exploration of new areas in the search space. Then one or more solutions from the current and/or from earlier iterations are used to increase the values of pheromone trail parameters on solution components that are part of these solutions.

As a prominent example, we outline in the following the pheromone update rule that was used in Ant System (AS) [66, 67], which was the first ACO algorithm proposed. This update rule is defined by

$$\tau_i \leftarrow (1-\rho)\cdot\tau_i + \rho\cdot\sum_{\{\mathfrak{s}\in\mathfrak{P}_{iter}|c_i\in\mathfrak{s}\}} F(\mathfrak{s}), \tag{3.1.2}$$

for $i = 1,\ldots,n$, where $\mathfrak{P}_{iter}$ is the set of solutions that were generated in the current iteration. Furthermore, $\rho \in (0,1)$ is the parameter called evaporation rate, and $F : \mathfrak{P} \to \mathbb{R}^+$ is a function such that $f(\mathfrak{s}) < f(\mathfrak{s}') \Rightarrow F(\mathfrak{s}) \geq F(\mathfrak{s}'), \forall \mathfrak{s} \neq \mathfrak{s}' \in \mathfrak{P}$.

Generally ACO algorithms mainly differ from each other by the way they update the pheromone values. See section 3.3.1.2 for the description of pheromone updating rules used in some well-known ACO algorithms like Ant Colony System, Best Worst Ant System, Max-Min Ant System.

**DaemonActions:** Daemon actions can used to implement centralized actions which cannot be performed by single ants. Examples are the application of local search methods to the constructed solutions, or the collection of global information that can be used to decide whether it is useful or not to deposit additional pheromone to bias the search process from a non-local perspective. As a practical example, the daemon may decide to deposit extra pheromone on the solution components that belong to the best solution found so far.

## 3.2 An Extended Version of Graph-based Ant System, its Applicability and Convergence

In this section we propose a generalized version of Gutjahr's Graph-based Ant System (GBAS) framework for solving static combinatorial optimization problems and also conduct an analysis of convergence properties of this generalized framework. A new transition rule, intended to balance between the exploration and the exploitation in the search progress of Ant-based algorithms, is incorporated into Gutjahr's GBAS model. As shown later, our generalized framework still holds all convergent properties of the GBAS model and probably show a promising improvement in the quality of solutions for Ant-based algorithms found in literature.

### 3.2.1 Introduction

Although many empirical studies have been conducted on ACO algorithms so far, very little has been reported on the theoretical convergence analysis

of this metaheuristic [62]. Below achievements are such recent attempts in theoretical analysis on probabilistic convergence for or runtime of ACO:

1. Gutjahr [106] proved convergence properties of an Ant System-based framework he proposed for static combinatorial optimization problems. That framework is called Graph-Based Ant System (GBAS). His finding about the convergence properties of the Ant-based framework is the first well-known theoretical result on the soundness of the Ant-based optimization approach. Under strong conditions, he proved that GBAS's current best solutions will converge to the global optimal solution of a given problem instance *with a probability made arbitrarily close to one*. The convergence properties of GBAS are theoretically valuable but, unfortunately, not practically important. The framework does not model closely enough any implemented ACO algorithms and this limits its range of applications. Moreover, although the convergent probability of GBAS can be made bigger by a reasonable value of either the evaporation factor or the number of agents, the GBAS's performance has not been evaluated yet. As GBAS model assumes that the construction graph is time-independent, i.e. the graph and the mapping function is unchanged when solving problems, therefore the GBAS is applicable only to static COPs.

2. Since GBAS is different from other ACO implementations, the application range of its convergent result is quite restrictive. Stützle and Dorigo [189] presented a class of ACO algorithms which are regarded as variants of Max-Min Ant System ( [190]). By adopting Gutjahr's proof method for GBAS into their proposed class of algorithms, Stützle and

Dorigo showed that the probability of obtaining an optimal solution at least once tends to one when the number of iterations approaches infinity. The algorithms in this class are stronger than GBAS in the sense of having a less restrictive application range; however, its convergence property is almost as weak as that for a random search. Another limitation of Stützle and Dorigo's convergence result is that the lower bound of the probability for the current solutions to be optimal solutions cannot be completely closed to one.

3. Recently, to overcome limitations of in GBAS and Stützle and Dorigo's framework, a time-dependent modification of GBAS along with its convergence result was investigated by Gutjahr [107]. With this modified GBAS framework, he showed that the current best solutions converge to an optimal solution *with a probability exactly equal to one*. This framework is practically better (but more complicated to implement) than GBAS. Additionally, the proposed framework has a stronger convergence property than that proposed by Stützle and Dorigo does. But there exists a disadvantage of this time-dependent framework which comes from tight conditions on a class of in-question problems and hence practical algorithms developed from the framework may be less efficient.

4. Recently, in an attempt to enlarge the applicability of GBAS, some tight conditions on a class of problems to tackle were removed in a more general model of GBAS suggested by Gutjahr [108]. Those tight conditions in GBAS are the condition on the uniqueness of the optimal solution

and that on the unique way of encoding this optimal solution. By relaxing these two conditions, Gutjahr showed that convergence properties of the original GBAS remain intact for the relaxed GBAS model. Additionally, he suggested that the condition on requiring a specific parameter update strategy at the first cycle can be relaxed by starting that strategy at a certain cycle. Although statements of convergence properties for the tight-conditions-relaxed model in [108] are the same as those for GBAS, findings for this relaxed model are actually more general and stronger than those for GBAS.

5. Most recently, a runtime analysis on a special ACO algorithm of 1 ant is conducted [60]. However, this analytic study is to focus on analyzing runtime of ant algorithm under a specific condition and not to study probabilistic convergence of ant algorithm which is the main focus of this thesis.

Despite playing an important role in improving solution quality, studying a so-called trade-off mechanism ( [65] and Part IV in [72]) has just stopped at empirical works and has not been found in any theoretical works so far. This mechanism is used in almost all implemented ACO algorithms, firstly in ACS [65] in which it defines the *exploitation-based exploration* of the search process[2]. In these implementations, a strategy of using a fixed positive value of a systematic parameter that is called *exploiting parameter* is applied to the mechanism. The trade-off mechanism can be simply ignored in an ACO algorithm by setting the value of this exploiting parameter to zero. Because of the absence of the mechanism in the studied frameworks, findings of the-

[2]This mechanism is considered as a pattern of intensification technique.

oretical works in the literature cannot explain the practically important role of the mechanism. In this paper, we aim at investigating convergence properties of an Ant-based framework that includes the trade-off mechanism by extending GBAS. The extended GBAS version is obtained by modifying the transition rule of GBAS such that the function of this tradeoff mechanism is modeled into that rule. Since our framework is different from Ant System-based frameworks described in [106–108, 189], clearly the results obtained in those works do not cover ours. It is also worthy of notice that investigation of an Ant-based time-dependent framework in which the trade-off mechanism is modeled is not in the scope of this present paper. The purpose of this paper is to theoretically examine the influence of the mechanism over the convergence of Ant-based algorithms. Because of GBAS's simplicity and its convergence properties evaluated, we choose GBAS as the original framework for this purpose. This section is organized as follows. The next subsection 3.2.2 is devoted to descriptions of the GBAS and our extended GBAS version (EGBAS for short). In subsection 3.2.3, the validity of convergence properties of EGBAS is proved. The last subsection will discuss about the importance of our findings.

### 3.2.2 A Generalized GBAS Framework

Since the framework we study is an extended variant of GBAS, thus in this subsection, first we recall Gutjahr's GBAS framework presented in [106], then describe the EGBAS proposed by us. Conceptual definitions and descriptions related to the GBAS framework are quoted with comments if and when deemed necessary.

#### 3.2.2.1 Graph-Based Ant Systems - GBAS

The GBAS uses a type of graph named *construction graph* whose definition is given below.

**Definition 3.2.1.** *Given an instance of a CO problem which is assumed as a minimization problem, a construction graph for this instance refers to a directed graph* $\mathcal{G} = (\mathcal{V}, \mathcal{A})$[3] *together with a function* $\Phi$ *with the following properties:*

*(1) In* $\mathcal{G}$*, there is only one node marked as the start node.*

*(2) Let* $\mathcal{W}$ *be the set of (directed) walks w in* $\mathcal{G}$ *satisfying the conditions: (i) w starts at the start node of* $\mathcal{G}$*, (ii) w contains each node of* $\mathcal{G}$ *at most once, and (iii) the last node on w has no successor node in* $\mathcal{G}$ *that is not already contained in w.*

*(3)* $\Phi$ *maps the set* $\mathcal{W}$ *onto a set of* $\mathcal{S}$ *containing all feasible solutions of the given problem instance. In other words, to each walk w satisfying (i)-(iii), there corresponds (via* $\Phi$*) a solution in* $\mathcal{S}$*, and to each solution in S (in particular: to each feasible solution), there corresponds (via* $\Phi^{-1}$*) at least one walk satisfying (i)-(iii).*

Thus, a specific encoding scheme of the feasible solutions as "walks" is clearly determined in a construction graph $(\mathcal{G}, \Phi)$. And the objective function value of a walk is set the same as that of its corresponding solution in $\mathcal{S}$.

Thus, with this encoding scheme, the search process will seek the "best" walks with the best value of the objective function in the $\mathcal{W}$ space instead of searching best solutions with the best value of the objective function in the $\mathcal{S}$ space. A graphic demonstration showing functional maps from the walks space to solution space via objective function is given in Fig. (3.1).

[3] $\mathcal{A}$ is the set of arcs in $\mathcal{G}$.

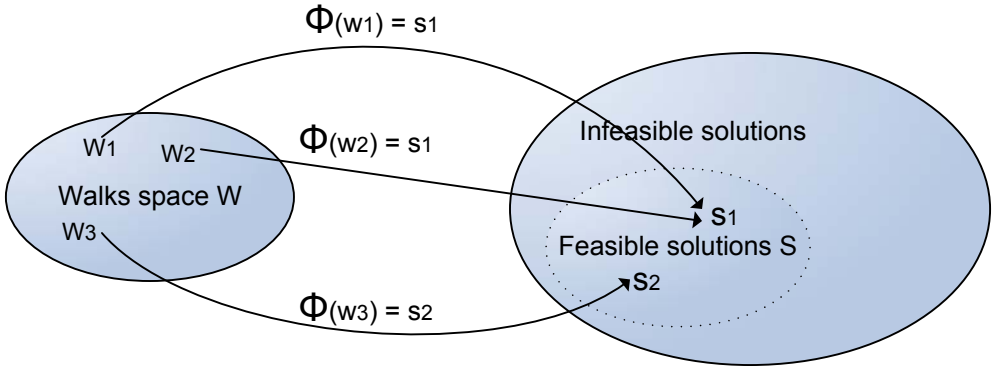


Figure 3.1: Functional relationship through the map $\Omega$ between walks space$\mathcal{W}$ and solutions space $\mathcal{S}$. $\{w_1, w_2\} \in \Omega^{-1}(s_1)$.

With the definition of a construction graph above, we will briefly re-state the components of the GBAS ( [106] for more details) below.

1. A construction graph $(\mathcal{G}, \Phi)$ according to the definition (3.2.1).

2. A set $A_1, \ldots, A_S$ of *agents*(most papers in literature called *ants*). Each agent performs a random walk with certain transition probabilities (see component 3 below) in $(\mathcal{G}, \Phi)$. The walk performed by each agent may be done on a separate processor if a multi-processor system is used, whereas, in a single-processor realization, moves of the agents are computed sequentially. An interval of time in which each agent carries out a walk (including many single moves) through $\mathcal{G}$ will be called a *cycle*. An implementation of AS has $M$ cycles; the number of cycles $M$ is possibly fixed in advance or decided at a later time during the execution of the algorithm.

3. *Transition probability* for each random move of the agents in each cycle

is defined as the following. Let $u = (u_0, \ldots, u_{t-1})$ indicate a partial walk that an agent has already traversed right before its $t^{th}$ transition step in a fixed cycle $m$, where $u_0, \ldots, u_{t-1}$ are node indices in $\mathcal{G}$ ($u_0$ referring to the start node). If node $l$ is visited in the partial walk $u$ then one may write $l \in u$, and $l \notin u$ otherwise. The following general form of the transition probabilities is used in almost every ACO algorithm,

$$p_{kl}(m,u) = \begin{cases} \dfrac{[\tau_{kl}(m)]^{\alpha}[\eta_{kl}(u)]^{\beta}}{\sum\limits_{r \notin u,(k,r)\in\mathcal{A}} [\tau_{kr}(m)]^{\alpha}[\eta_{kr}(u)]^{\beta}}, & \text{if } l \notin u \text{ and } (k,l) \in \mathcal{A} \\ 0, & \text{otherwise} \end{cases}, \quad (3.2.1)$$

where $p_{kl}(m,u)$ indicates the probability that a fixed agent having already traversed a partial walk $u = (u_0, \ldots, u_{t-1} = k)$ in the current cycle $m$, moves from the current node $k$ to node $l$. It should be noted that this probability is only determined if $k = u_{t-1}$. The numbers $\tau_{kl}(m)$ are so-called "pheromone values" (see component 4 below), and the numbers $\eta_{kl}(m)$ are called "desirability values" (see component 5 below). $\alpha$ and $\beta$ are parameters. As presented later, *the mechanism of balancing between the exploitation and the exploration in search progress is incorporated into this component of GBAS forming our EGBAS framework.*

4. An array of *pheromone values* $\tau_{kl}$ assigned to arc $(k,l)$ in $(\mathcal{G}, \Phi)$. Because the pheromone values change from cycle to cycle (see below), we can represent their dependence on the cycle index $m$ in the form $\tau_{kl}(m)$. Let $\tau_{kl}(1)$ be initialized to 1/(number of arcs) $\forall (k,l) \in \mathcal{A}$. At the end of each cycle $m$, the following pheromone updating rule is carried out. For each agent $A_s$ and each arc $(k,l)$, a value $\Delta\tau_{kl}^{(s)}$ is defined as a function of the solution assigned to the walk of $A_s$ in the current cycle $m$. Suppose this

solution has a cost value (objective function value) $f_s$ then for each arc $(k, l)$,

$$\Delta\tau_{kl}^{(s)} = \begin{cases} \varphi(f_s), & \text{if } A_s \text{ has travesed arc } (k,l), \\ 0, & \text{otherwise,} \end{cases} \tag{3.2.2}$$

where $\varphi$ is a non-increasing function which may depend on the walks of the agents in the cycles $1, \ldots, m-1$. Let

$$\mathcal{C} = \sum_{(k,l)\in A} \sum_{s=1}^{S} \Delta\tau_{kl}^{s}. \tag{3.2.3}$$

If $\mathcal{C} = 0$, we set

$$\tau_{kl}(m+1) = \tau_{kl}(m) \; \forall (k,l). \tag{3.2.4}$$

If, otherwise, $\mathcal{C} > 0$, we set

$$\tau_{kl}(m+1) = (1-\rho)\tau_{kl}(m) + \rho\Delta\tau_{kl}, \tag{3.2.5}$$

where[4]

$$\Delta\tau_{kl} = \frac{1}{\mathcal{C}} \sum_{s=1}^{S} \Delta\tau_{kl}^{(s)}. \tag{3.2.6}$$

5. An array of *desirability values* $\eta_{kl}$ is assigned to arcs $(k, l)$ in $\mathcal{G}$. The desirability values possibly depend on the partial walk $u = (u_0, \ldots, u_{t-1} = k)$ that the current agent has already traversed, so they can be written as $\eta_{kl} = \eta_{kl}(u)$. Typically, the value $\eta_{kl}(u)$ is obtained from a certain *Greedy Heuristic* applicable to the problem considered.

#### 3.2.2.2 Extension of GBAS - EGBAS

As mentioned earlier in 3.2.1, a tradeoff mechanism is added into the transition rule of GBAS forming the proposed framework. The only difference

[4]The number $\rho$ is called the *evaporation factor* (see [67]).

between our EGBAS and GBAS is at the transition procedure and described as follows.

Let $q_0 \in [0, 1]$ be a systematic constant called the exploiting parameter. A random number $q \in [0, 1)$ is always produced before any agent makes a move to the next node. Let $u = (u_0, \ldots, u_{t-1} = k)$ be the partial walk which the agent has already traversed at cycle $m$. Then, its next node $\mathcal{L}$ is chosen according to the below equation (3.2.7):

$$\mathcal{L} = \begin{cases} \arg \max\limits_{r \notin u,\ (k,r) \in \mathcal{A}} \{[\tau_{kl}(m)]^{\alpha}[\eta_{kl}(u)]^{\beta}\}, & \text{if } q \leq q_0 \\ l, & \text{otherwise,} \end{cases} \tag{3.2.7}$$

where the node $l$ is chosen according to ( 3.2.1 ). It is clear that by setting $q_0 = 0$, our model becomes GBAS. In other words, GBAS is a special case of the proposed model.

Notice that (3.2.7 is the equation used to represent for the trade-off mechanism in ant algorithms in their transition rule.

Thus, in GBAS the next move of any agent is picked up according to ( 3.2.1 ), in EGBAS whereas it is according to ( 3.2.7 ).

**Remark 3.2.1.** *The incorporation of the balance mechanism into GBAS does not affect the "structure" of the applicable problems in GBAS's application range at all. Therefore the application range of the EGBAS framework remains the same as that of GBAS. It should be noted that the EGBAS is applicable to every combinatorial optimization problem with a finite solution space the same as GBAS.*

*Given an instance of such a problem, a simple construction graph can have only a start node, a termination node, a specific node $v_x$ for each feasible node $x$, and arcs from the start node to any $v_x$ and from each $v_x$ to the termination node. This graph is*

*obviously not practical, but we show it as an example of set of applicable construction graphs.*

### 3.2.3 Convergence of EGBAS

We now show that the convergence property of GBAS is still valid in our framework EGBAS. The convergence property of EGBAS, that will be stated in theorem ( 3.2.2 ) and be proven in this subsection, is informally restated as follow: "under some conditions, the current solutions of EGBAS converge to the optimal solution with the probability which is as arbitrarily close to 1 as we want". Such conditions are restated as follows [106][5].

a. The parameter $\alpha$ in ( 3.2.1 ) is selected as $\alpha = 1$.

b. There is only one optimal walk in $\mathcal{W}$. It is encoded by only one walk in $\mathcal{W}$.

c. Along the optimal walk $w^*$, the desirability value satisfies $\eta_{kl}(u) > 0 \; \forall (k,l)$ either $\in w^*$ or corresponding partial walks $u$ of $w^*$.

d. Let $f^* = f^*(m)$ be the lowest cost value observed in the cycles $1, \ldots, m-1$, that is, the lowest objective function value $f_s$ corresponding to a walk of an agent $A_s$ in these $m-1$ cycles. At $m = 1$, let $f^* = \infty$. Assume the function $\varphi$ chosen for the definition of the values $\Delta\tau_{kl}^{(s)}$ at the beginning of cycle $m+1$ (see ( 3.2.2 )) has the below properties:

   i. $\varphi(f_s) > 0$ for $f_s \leq f^*$ and

[5]Interested reader may refer to [106] for more explanations on these conditions.

ii. $\varphi(f_s) = 0$ for $f_s > f^*$.[6]

Since our purpose is on investigating the convergence property of EGBAS, we will show that the only theorem related to convergence properties of GBAS (Theorem 4.1 in [106]) is still valid for EGBAS without changing its statement. The theorem is restated below.

**Theorem 3.2.2.** *Let conditions (a)-(d) be satisfied, and $P_m$ denote the probability that a fixed agent traverses the optimal walk in cycle $m$. Then the following two assertions are valid:*

1. *For each $\epsilon > 0$ and for fixed parameters $\rho$ and $\beta$, it can be achieved by the choice of a sufficiently large number $S$ of agents that $P_m \geq 1 - \epsilon$ holds $\forall m \geq m_0$ (with an integer $m_0$ depending on $\epsilon$).*

2. *For each $\epsilon > 0$ and for fixed parameters $S$ and $\beta$, it can be achieved by the choice of an evaporation factor $\rho$ sufficiently close to zero that $P_m \geq 1 - \epsilon$ holds $\forall m \geq m_0$ (with an integer $m_0$ depending on $\epsilon$).*

The search procedure of EGBAS is considered as a *Markov process* in discrete time if the states of this Markov process are formed as the triple

$$(\underline{\tau}(m), \underline{w}(m), f^*(m)) \ (m = 1, 2, \ldots),$$

where

- $\underline{\tau}(m)$ is the vector of the pheromone values $\tau_{kl}(m)$ $\forall(k, l)$ during cycle $m$.
- $\underline{w}(m)$ is the vector of the walks $w^{(s)}$ $(s = 1, \ldots, S)$ of the agents $A_1, \ldots, A_S$ in cycle $m$.

[6]It means that only walks which are at least as good as the best-found-so-far walk receive a positive increment $\Delta_{kl}^{(s)}$.

- $f^*(m)$ is the best found cost value corresponding to the walk of any agent in cycle $1, \ldots, m-1$, and $f^*(1) = \infty$.

**Proposition 3.2.3.** *[Proposition 1 in [106]] The state variables ($\underline{\tau}(m)$, $\underline{w}(m)$, $f^*(m)$) (m=1,2,...) form a Markov process.*

It is clear that the addition of the tradeoff mechanism into GBAS's transition probability does not affect the Markov characteristics of GBAS, thus, this proposition is still valid in EGBAS.

Before going further into the details of proofs, let us define the symbols that are used in those proofs (all these symbols are used with the same notions and meanings as in and restated from [106]).

- $w^*$ indicates the optimal walk.
- $L$ denotes the length (number of arcs) of $w^*$.
- $Pr$ is the probability measure on the Markov process defined above.
- $E_m^{(s)}$ indicates the event $w^{(s)}(m) = w^*$.
- $B_m$ stands for $\neg E_m^{(1)} \wedge \ldots \wedge \neg E_m^{(S)}$.
- $F_m$ stands for $B_1 \wedge \ldots \wedge B_{m-1} \wedge \neg B_m$, while $F^*$ stands for $F_1 \vee F_2 \vee \ldots$.

The following extra symbols are needed to prove the convergence in the case of EGBAS. Let

$$u_i = (u_0, v_1, v_2, \ldots, v_i)$$

be a partial walk of $w^*$, then

- $X_{v_i r}(m, u_i)$ be the event of $\left\{\tau_{v_i v_{i+1}} \eta^{\beta}_{v_i v_{i+1}} > \tau_{v_i r} \eta^{\beta}_{v_i r}\right\}$, where $r \notin u_{i+1}, (v_i, r) \in \mathcal{A}$.

- $P_i(m, u_i) = \bigwedge_{r \notin u_{i+1}, (v_i, r) \in A} X_{v_i r}(m, u_i)$.

According to ( 3.2.7 ), at any time an agent makes a move, the node $v_{i+1}$ is picked up as that agent's next node with a probability $Pr\{P_i(m, u_i)\}$ if $q \leq q_0$. So, a fixed agent will randomly pick up node $v_{i+1}$ as its next node while standing at current node $v_i$ (with the current partial walk $u_i$) at cycle $m$ with the the following probability:

$$P_{v_i v_{i+1}}(m, u_i) = q_0 p_{v_i v_{i+1}}(m, u_i) + (1 - q_0) Pr\{P_i(m, u_i)\}, \tag{3.2.8}$$

where $p_{v_i v_{i+1}}(m, u_i)$ is computed in ( 3.2.1 ). For the sake of simplicity in expressions, we use $P_i(m, u_i)$ from this point onward instead of $Pr\{P_i(m, u_i)\}$ unless otherwise stated. It is worth mentioning here, for a cautious purpose, the difference of meaning of notions that are $P_{v_i v_{i+1}}$ (uppercase $P$) and $p_{v_i v_{i+1}}$ (lowercase $p$). The latter means the state transition probability of GBAS (without the presence of the trade-off mechanism), whereas the former is that of EGBAS.

By setting $c_{v_i r} = \left[ \dfrac{\eta_{v_i v_{i+1}}(u_i)}{\eta_{v_i r}(u_i)} \right]^{\beta}$, which may be seen as a constant, then

$$Pr\{X_{v_i r}(m, u_i)\} = Pr\left\{ \frac{\tau_{v_i r}(m)}{\tau_{v_i v_{i+1}}(m)} < c_{v_i r} \right\} \quad \forall r \notin u_{i+1}, (v_i, r) \in \mathcal{A}, \tag{3.2.9}$$

and,

$$Pr\{P_i(m, u_i)\} = \prod_{r \notin u_{i+1}, (v_i, r) \in \mathcal{A}} Pr\{X_{v_i r}(m, u_i)\}. \tag{3.2.10}$$

It can be assumed without loss of generality that

$$\gamma = \min \left\{ \eta_{kl}^{\beta}(u) | (k, l) \in w^*, \ u \text{ is any partial walk of } w^* \right\} > 0, \tag{3.2.11}$$

and $\Gamma = \max \left[ \eta_{kl}^{\beta}(u) \right] < \infty$. Then, by normalizing, we obtain

$$\left[ \eta_{kl}^{\beta}(u) \leq 1 \right], \tag{3.2.12}$$

for all arcs $(k, l) \in \mathcal{A}$ and all partial walks $u$.

Our strategy of proving theorem (3.2.2) is firstly to prove that lemmas corollaries stated in [106] are still valid in the EGBAS context. Then, due to the consequence of modifying the transition rule of GBAS, the proof of theorem (3.2.2) may require other supplemental results, thus the strategy will also consists of seeking and proving such supplemental results. If and when necessary, the supplement results will be highlighted. Moreover, we adopt the proof technique from Gutjahr's proof in [106] to prove theorem (3.2.2) (in the EGBAS context).

The next section will recall necessary results (lemmas and corollaries) used to prove theorem (3.2.2) (in the context of GBAS). Our proof for this theorem but in the new context - EGBAS - still rely on these results (in EGBAS context), so we need to prove that they are still valid in EGBAS context. Their proofs and supplemental results are mentioned in the section 3.2.3.1.

#### 3.2.3.1 Convergence of EGBAS

This section is devoted to present proofs for lemmas and corollaries used to proof theorem ( 3.2.2 )[7]. It also presents supplemental results and their proofs if and when deemed necessarily.

**Lemma 3.2.4.** *[similar to Lemma (C.0.1)] The probability $Pr(\neg B_m)$ which at least one agent traverses the optimal walk in cycle m is not less than $1 - (1 - c^{m-1}p)^S$, where $c = (1-\rho)^L$ and $p = q_0^L \gamma^L \cdot \prod_{(k,l)\in w^*} \tau_{kl}(1)$ with $\gamma$ defined by ( 3.2.11 ).*

[7]Similar lemmas and corollaries to these and used in the proof of a similar theorem in GBAS's context can be found in the appendix of this thesis.

This Lemma is a bit different from the Lemma (C.0.1) by the presence of $q_0$ in the last equation of computing $p$.

*Proof.* From (3.2.4), (3.2.5), and (3.2.6), it is clear that

$$\begin{aligned} \tau_{lk}(m+1) &\geq (1-\rho)\tau_{kl}(m) \\ &\dots \\ &\geq (1-\rho)^m \tau_{kl}(1). \end{aligned} \tag{3.2.13}$$

Due to (3.2.11), (3.2.12) and the method of updating $\underline{\tau}(m)$, then

$$\sum_{r \notin u,\ (k,r)\in\mathcal{A}} \tau_{kr}(m)\eta_{kr}^{\beta}(u) \leq 1,$$

so

$$\begin{aligned} p_{kl}(m,u) &= \frac{\tau_{kl}(m)\eta_{kl}^{\beta}(u)}{\sum\limits_{r \notin u,\ (k,r)\in\mathcal{A}} \tau_{kr}(m)\eta_{kr}^{\beta}(u)} \\ &\geq \tau_{kl}(m)\eta_{kl}^{\beta}(u) \geq \gamma\tau_{kl}(m). \end{aligned} \tag{3.2.14}$$

Let $w^* = (u_0, v_1, v_2, \dots, v_L)$. From (3.2.8), (3.2.13), (3.2.14),

$$\begin{aligned} Pr(E_m^{(s)}) &= \prod_{i=0}^{L-1} P_{v_i v_{i+1}}(m,u_i) = \prod_{i=0}^{L-1} \{q_0 p_{v_i v_{i+1}}(m,u_i) + (1-q_0)P_i(m,u_i)\} \\ &\geq \prod_{i=0}^{L-1} q_0\gamma\tau_{v_i v_{i+1}}(m) \geq q_0^L\gamma^L \prod_{i=0}^{L-1}(1-\rho)^{m-1}\tau_{v_i v_{i+1}}(1) = c^{m-1}p. \end{aligned}$$

Since the walks of $S$ agents are independent mutually, this implies

$$Pr(\neg B_m) \geq 1 - (1 - c^{m-1}p)^S.$$

□

Similar to the proof of Lemma (3.2.4), the following corollary is obtained:

**Corollary 3.2.5.** *[similar to Corollary C.0.2] The conditional probability* $Pr(\neg B_m | B_1 \wedge \dots \wedge B_{m-1}) \geq 1 - (1 - c^{m-1}p)^S$.

Herein, in the following lemmas and corollaries, assertions on conditional probabilities which are conditional on event $F_m$ shall be formed.

Proof of Lemma (C.0.3) in EGBAS context:

*Proof.* By considering two cases concerning to values of $\mathcal{C}$ (computed in ( 3.2.3 )), one case is $\mathcal{C} = 0$ and the other $\mathcal{C} > 0$, it is not difficult to obtain, with $m' > m$,

$$\tau_{kl}(m' + 1) \geq \min\{\tau_{kl}(m + 1), 1/L\} \geq \alpha_{kl}(m),$$

where $\alpha_{kl}(m) = (1 - \rho)^m \tau_{kl}(1)$.

Let $(k, l) \in w^*$, and $u = (u_0, \ldots, u_{t-1} = k)$ be the partial walk from the start node to node $k$ on $w^*$. By ( 3.2.14 ), we have

$$p_{kl}(m', u) \geq \gamma \tau_{kl}(m') \geq \gamma \alpha_{kl}(m).$$

Thus, for a fixed agent $A_s$ and $m' > m$,

$$\begin{aligned} Pr(E^{(s)}_{m'}|F_m) &\geq \prod_{(k,l)\in w^*} q_0 p_{kl}(m', u) \\ &\geq \prod_{(k,l)\in w^*} q_0 \gamma \alpha_{kl}(m) \\ &= [q_0\gamma]^L \prod_{(k,l)\in w^*} \alpha_{kl}(m) \\ &= a(m) > 0, \end{aligned} \tag{3.2.15}$$

It is obvious that the number $a(m)$ is independent of $m'$. This result, in turn, shows that even though there exists a difference between GBAS's transition probability and our EGBAS, the remaining part of proof of Lemma 4.2 in [106] is left unchanged[8], or Lemma (C.0.3) is proved. □

[8]The only difference between this proof and the corresponding in [106] is the appearance of $q_0$ in ( 3.2.15 ).

**Corollary 3.2.6.** *For each $\epsilon > 0$ and each $m \in N$ there is an integer $d'(\epsilon, m)$ such that $\forall\ m' \geq m + d'(\epsilon, m)$ then*

$$Pr\{P_i(m', u_i) \geq 1 - \epsilon | F_m\} \geq 1 - \epsilon. \tag{3.2.16}$$

*Proof.* From Lemma (C.0.3) (in EGBAS context), it clearly brings the following result to us:

"For each $\widetilde{\epsilon} > 0$ and each $\widetilde{m} \in N$, there exists an integer $\widetilde{d}(\widetilde{\epsilon}, \widetilde{m}) \in N$ such that

$$Pr\left\{\frac{\tau_{kr}(m')}{\tau_{kl}(m')} \leq \widetilde{\epsilon} | F_{\widetilde{m}}\right\} \geq 1 - \widetilde{\epsilon},$$

$\forall (k, l) \in w^*, (k, r) \notin w^*$ and $\forall m' \geq \widetilde{m} + \widetilde{d}(\widetilde{\epsilon}, \widetilde{m})$."

So, $\forall r \notin u_{i+1}, (v_i, r) \in A$ and $\forall \widetilde{\epsilon} \leq \min\limits_{r}\{c_{v_i r}\}$ where $c_{v_i r}$ defined in ( 3.2.8 ), and $\forall m' \geq \widetilde{m} + \widetilde{d}(\widetilde{\epsilon}, \widetilde{m})$, then

$$Pr\left\{\frac{\tau_{v_i v_{i+1}}(m')}{\tau_{v_i r}(m')} \leq c_{v_i r} | F_{\widetilde{m}}\right\} \geq Pr\left\{\frac{\tau_{kr}(m')}{\tau_{kl}(m')} \leq \widetilde{\epsilon} | F_{\widetilde{m}}\right\} \geq 1 - \widetilde{\epsilon}. \tag{3.2.17}$$

Hence, from the definition of $P_i(m, u_i)$ and taking a constant integer $M = card\{r \notin u_{i+1}, (v_i, r) \in A\}$, we have with a probability of at least $1 - \widetilde{\epsilon}$,

$$\begin{aligned} P_i(m', u_i) &= \prod_{r \notin u_{i+1}, (v_i, r) \in A} Pr\left\{\frac{\tau_{v_i v_{i+1}}(m')}{\tau_{v_i r}(m')} \leq c_{v_i r} | F_{\widetilde{m}}\right\} \\ &\geq \prod_{r \notin u_{i+1}, (v_i, r) \in A} (1 - \widetilde{\epsilon}) \\ &= (1 - \widetilde{\epsilon})^M. \end{aligned}$$

By choosing $\epsilon$ such that[9] $\widetilde{\epsilon} = 1 - (1 - \epsilon)^{1/M}$, there is an integer $d'(\epsilon, m)$ such that $\forall\ m' \geq m + d'(\epsilon, m)$

$$Pr\left\{P_i(m', u_i) \geq (1 - \widetilde{\epsilon})^M = 1 - \epsilon | F_{\widetilde{m}}\right\} \geq 1 - \widetilde{\epsilon} \geq 1 - \epsilon.$$

[9]Since $\widetilde{\epsilon}$ can assume any sufficiently small value, thus so can $\epsilon$.

The corollary is proven. □

Lemma (C.0.4) (Lemma 4.3 in [106]) still holds (in EGBAS context) since components in its proof do not have any relationship to the parameter $q_0$, and what its proof needs is Lemma (C.0.3) which is already proved above. An extended version of Lemma (C.0.4) for EGBAS case is the following:

**Lemma 3.2.7.** *Lemma (C.0.4) above still holds if the quantity $p_{kl}(m', u^*(k))$ is replaced by $P_{kl}(m', u^*(k))$ defined in ( 3.2.8 ), i.e. for each $\epsilon > 0$ and each $m \in N$ there is an integer $d'''(\epsilon, m) \in N$ such that $\forall (k,\ l) \in w^*$ and $\forall m' \geq m + d'''(\epsilon, m)$*

$$Pr\{P_{kl}(m', u^*(k)) \geq 1 - \epsilon | F_m\} \geq 1 - \epsilon.$$

*Proof.* Given conditional probability on $F_m$, from Corollary 3.2.6 and Lemma (C.0.4) and the definition of $P_{kl}(m', u^*(k))$ in ( 3.2.8 ), with a probability at least $1 - \epsilon$, we have:

- An integer $d'(\epsilon, m)$ satisfying: $\forall\ m' \geq m + d'(\epsilon, m)$, $P_i(m', u_k^*) \geq 1 - \epsilon$.
- An integer $d''(\epsilon, m)$ satisfying: $\forall\ m' \geq m + d''(\epsilon, m)$, $p_{kl}(m', u^*(k)) \geq 1 - \epsilon$.

Hence, with a probability at least $1-\epsilon$, $P_{kl}(m', u^*(k)) \geq q_0(1-\epsilon)+(1-q_0)(1-\epsilon) = 1 - \epsilon$ for all $m' \geq m + \max\{d'(\epsilon, m), d''(\epsilon, m)\} = m + d'''(\epsilon, m)$. Thus the Lemma is proved. □

**Corollary 3.2.8.** *[similar to Corollary C.0.5] With the notations in Lemma (3.2.7), set*

$$Y_{m'} = \prod_{(k,l) \in w^*} P_{kl}(m', u^*(k)). \tag{3.2.18}$$

*Then, for each $\epsilon > 0$ and each $m \in N$, there is an integer $d'''(\epsilon, m) \in N$ such that*

$$Pr\{Y_{m'} \geq 1 - \epsilon | F_m\} \geq 1 - \epsilon.$$

*for all* $m' \geq m + d'''(\epsilon, m)$.

The difference between this corollary and Corollary C.0.5 is that the quantity $p_{kl}(m', u^*(k))$ in the latter is replaced by $P_{kl}(m', u^*(k))$ in the former.

*Proof.* By following the same steps used in Corollary C.0.5's proof (in [106]) but replacing symbol $p_{kl}(m', u^*(k))$ with $P_{kl}(m', u^*(k))$ and the notion of Lemma (C.0.4) with that of Lemma (3.2.7) we obtain this corollary. □

The next is the proof of Lemma (C.0.6) in EGBAS context.

*Lemma (C.0.6).* Similar to the way used to prove Corollary 3.2.8, we prove this Lemma in the context of EGBAS by following the proof of Lemma (C.0.6) (Lemma 4.4 in [106]) step by step but remember to replace the notions: *Corollary C.0.5* with *Corollary 3.2.8* and *Lemma (C.0.4)* with *Lemma (3.2.7)*. □

Taking all the corollaries and lemmas described above, we now establish the proof of theorem (3.2.2).

*Proof of theorem (3.2.2).* We have shown that all Lemmas and Corollaries used in proof of Theorem 4.1 in [106] still hold for the case when the probabilistic transition rule of GBAS is replaced with that of EGBAS as given in ( 3.2.8 ). To prove theorem (3.2.2), one just needs to follow the steps in [106] but replacing symbols such as indices of Lemmas, Corollaries, Proposition, $A$ with $F^*$, and $p_{kl}(m, u(k))$ with $P_{kl}(m, u(k))$ if and when necessary. So, it can be shown that this theorem (Theorem 4.1 ) also holds for EGBAS (in other words, theorem (3.2.2) is proven completely).

However, to make the reading easier, appendix B will briefly present the proof of theorem (3.2.2) in terms of steps in Gutjahr's proof with necessary replacements of symbols and notions according to our model. □

**Remark 3.2.9.** *Theorem ( 3.2.2 ) states that by making the value of evaporation factor $\rho$ small enough and also keeping number of agents $S$ and $\beta$ constant, the probability that at least one agent visits the unique optimal solution $P_m$ can be arbitrarily close to 1 after certain number of run cycles m. In other words, probability associated convergence of EGBAS is arbitrarily close to 1 as we want.*

### 3.2.4 Discussion

Similar to Gutjahr's remarks about convergent results of GBAS, our results here also do not tell explicitly how to choose reasonable values of "number of agents" $S$ and/or "evaporation factor" $\rho$ in order to gain a high convergent probability in applications. But these convergent results show that the addition of the tradeoff mechanism into the GBAS framework does not change its convergence properties. As well as theoretical limits in GBAS, however, we do not know how the addition impacts the convergence speed of the EGBAS framework, and also how the convergent probability is affected by the exploiting parameter from the theoretical point of view. What we know is that values of this parameter is set very high in applications[10]. There should not be misled to statement that "the higher value of $q_0$, the better is the probability of convergence". As a consequence, the chance of selecting the next node based on ( 3.2.1 ) of a fixed agent is quite small (= $1 - q_0$). In addition, as we can see, GBAS is a special case of our model when $q_0$ is set to 0. From a theoretical point of view, GBAS could not explain the balancing mechanism's role in ACO-based applications, whereas EGBAS does this better. GBAS did not model the pseudo-random transition rule (aka the balancing mechanism)

[10]In ACS, $q_0 = 0.9$ for small size TSP instances, and $q_0 = 0.98$ for large ones.

also GBAS is a special model of that which is modeling the pseudo-random transition rule hence convergence properties obtained from GBAS cannot be adapted to algorithms using that pseudo-random transition rule. EGBAS's convergence properties can do so since EGBAS models the rule explicitly.

Another point to be noted here is that problems with infinite solution space such as those in continuous optimization area and some in dynamic optimization do not belong to the application scope of EGBAS. However, we find that it is still possible to apply the EGBAS to some of the continuous optimization problems by *discretizing their continuous solution space*[11]

## 3.3 Dynamically Updating the Exploiting Parameter in Improving Performance of Ant-based Algorithms

The utilization of *pseudo-random proportional transition* rule (or trade-off mechanism) to balance between the exploitation and exploration of the search process was proposed firstly in Ant Colony System (ACS) algorithm. This rule is, then, widely used in many Ant-based implementations. In this rule, a parameter of notion $q_0$ which is henceforth called *exploiting parameter*[12] defines the extensiveness of the trade-off exploitation-based exploration. However, in all

[11]By discretizing, the final solutions are certainly near-optimal in applying to the continuous case. But it is the fact that the EGBAS itself is an heuristic method whose final solutions are not mathematically confirmed to be optimum even in discrete case.

[12]A description of notions and technical terms for this rule were also intensively discussed in section 3.2. For the sake of easily following for readers, we recall them here if and when deemed.

Ant-based implementations applied for TSP, this rule has been either omitted or applied with a constant value of $q_0$. Instances of those using the rule are Ant Colony System (ACS), Max-Min Ant System (MMAS), Rank-based Ant System (RAS), and Best-Worst Ant System (BWAS).

In ACS, this rule is governed by a parameter so-called exploiting parameter which is always set to a constant value. Besides, all ACO-based algorithms either omit this rule or apply it with a fixed positive value of the exploiting parameter during the runtime of algorithms.

Moreover, one of attempts to understand the role of this rule in the aspect of algorithm convergence was reported in section 3.2. Findings from that attempt about influence of this rule on convergence of Ant-based algorithms are the main motivation for an empirical studying about the behavior of those algorithms in which their exploiting parameter is dynamically tuned over runtime. To carry out this empirical investigation, we incorporate this dynamical updating trade-off rule into MMAS, ACS, BWAS algorithms when applied them to symmetric TSP benchmark instances. Details of how to dynamically adapt the value of $q_0$ in chosen ACO algorithms will be introduced in section 3.3.1 as well. The next section will be devoted to analyze and compare the performance of these modified algorithms with their original version (fixed value of $q_0$). Finally, some concluding remarks and future works will be mentioned in the last section 3.3.3.3.

### 3.3.1 Ant Colony Optimization for Traveling Salesman Problem

#### 3.3.1.1 Traveling Salesman Problem

The TSP is formally defined as: "Let $V = \{a_1, .., a_n\}$ be a set of cities where $n$ is the number of cities, $A = \{(r, s) : r, s \in V\}$ be the set of edges, and $\delta(r, s)$ be the cost measure associated with the edge $(r, s) \in A$. The objective is to find a minimum cost closed tour that goes through each city only once." In the case that all of cities in $V$ are given by their coordinates and $\delta(r, s)$ is the Euclidean distance between any $r$ and $s$ $(r, s \in V)$ then this is so-called an Euclidean TSP problem. If $\delta(r, s) \neq \delta(s, r)$ for at least one edge $(r, s)$ then TSP becomes asymmetric TSP (ATSP).

#### 3.3.1.2 ACO algorithms for TSP

A simplified framework of ACO is recalled from [64] in Alg. (15).

**Algorithm 15** Ant Colony Optimization for Traveling Salesman Problem

```
1: Initialize
2: while termination conditions not met do
3:    // at this level, each loop is called an iteration
4:    Each ant is positioned on a starting node
5:    while all ants haven't built a complete tour yet do
6:       Each ant applies a state transition rule to increasingly build a solution.
7:       Each ant applies a local pheromone updating rule.{optional}
8:    end while
9:    Apply the so-called online delayed pheromone trail updating rule.{optional}
10:   Evaporate pheromone.
11:   Perform the deamon actions. {optional: local search, global updating}
12: end while
```

Following ACO-based algorithms share the same general state transition

rule when they are applied to TSP. That is, at a current node $r$, a certain ant $k$ will make a move to a next node $s$ in terms of the following probability distribution:

$$p_k(r,s) = \begin{cases} \dfrac{[\tau_{rs}^{\alpha}] \cdot [\eta_{rs}^{\beta}]}{\sum_{u \in J_k(r)} [\tau_{ru}^{\alpha}] \cdot [\eta_{ru}^{\beta}]}, & \text{if } s \in J_k(r) \\ 0, & \text{otherwise} \end{cases}, \tag{3.3.1}$$

where $J_k(r)$ is the set of nodes which ant $k$ has not visited yet; $\tau_{rs}$ and $\eta_{rs}$ are respectively the pheromone value (or called trail value sometimes) and the heuristic information of the edge $(r, s)$. Brief descriptions of operation of ACS, BWAS, MMAS are shown next.

**ACS: Ant Colony System**

*Transition rule:* The next node $s$ is chosen as follows:

$$s = \begin{cases} \arg\max_{u \in J_k(r)} \{[\tau_{ru}]^{\alpha} \cdot [\eta_{ru}]^{\beta}\}, & \text{if } q \le q_0 \\ \mathbf{S}, & \text{otherwise} \end{cases}, \tag{3.3.2}$$

where $\mathbf{S}$ is selected according to ( 3.3.1 ), $q_0 \in [0,1]$ is the exploiting parameter, $0 \le q \le 1$ is a random variable.

*Local updating rule:* When an ant visits an edge, it modifies the pheromone of that edge in the following way[13]:

$$\tau_{rs} \leftarrow (1-\rho) \cdot \tau_{rs} + \rho \cdot \Delta\tau_{rs},$$

where $\Delta\tau_{rs}$ is a systematic parameter with a fixed positive value.

*Global updating rule:* This rule is done by the deamon procedure which allows only the best-so-far ant to update pheromone values[14].

[13] Another name is *online step-by-step updating rule*.

[14] It is sometimes called *off-line pheromone updating rule* in other studies.

**BWAS: Best-Worst Ant System**

*Transition rule:* of BWAS is based on only ( 3.3.1 ).

*Local updating:* Moreover it does not use online pheromone updating rule. The local updating as being used in ACS is discarded in BWAS.

*Global updating:* Adopting the idea from Population-Based Incremental Learning (PBIL) [10] of considering both current best and worst ants, BWAS allows these two ants to perform positive and negative global pheromone updating rules respectively according to ( 3.3.3 ) and ( 3.3.4 ).

$$\tau_{rs} \leftarrow (1-\rho)\cdot\tau_{rs} + \Delta\tau_{rs} \tag{3.3.3}$$

where

$$\Delta\tau_{rs} = \begin{cases} f(C(S_{\text{global-best}})), & \text{if } (r,s) \in S_{\text{global-best}} \\ 0, & \text{otherwise} \end{cases}$$

$f(C(S_{\text{global-best}}))$ is the amount of trail to be deposited by the best-so-far ant. The following equation showing how the worst ant performs pheromone updating:

$$\tau_{rs} \leftarrow (1-\rho)\cdot\tau_{rs} \quad \forall(r,s) \in S_{\text{current-worst}} \text{ and (r, s)} \notin \text{S}_{\text{global-best}}. \tag{3.3.4}$$

*Restart:* A restart of the search progress is done when it gets stuck.

*Introducing diversity:* BWAS also performs the "mutation" for the pheromone matrix to introduce diversity in the search process. Each component of pheromone matrix is mutated with a probability $P_m$ as follows:

$$\tau_{rs}^{'} = \begin{cases} \tau_{rs} + mut(it, \tau_{\text{threshold}}), & \text{if a = 0} \\ \tau_{rs} - mut(it, \tau_{\text{threshold}}), & \text{if a = 1} \end{cases}$$

$$\tau_{\text{threshold}} = \frac{\sum_{(r,s)\in S_{\text{global-best}}} \cdot \tau_{rs}}{|S_{\text{global-best}}|}$$

with $a$ being a binary random variable[15], $it$ being the current iteration, and $mut(\cdot)$ being:

$$mut(it, \tau_{\text{threshold}}) = \frac{it - it_r}{Nit - it_r} \cdot \sigma \cdot \tau_{\text{threshold}} \tag{3.3.5}$$

where $Nit$ is the maximum number of iterations and $it_r$ is the last iteration where a restart was done.

**MMAS: Max-Min Ant System**

*Transition rule:* of MMAS is the same as BWAS, e.g. it uses only ( 3.3.1 ) to choose the next node. A variant of MMAS also used the pseudo-random proportional transition rule [186, 191].

*Local updating:* The same as BWAS, no local updating rule is used in MMAS.

*Global updating:* After all ants complete their tours, pheromone trails on all arcs are evaporated according to the following equation:

$$\tau_{ij} \rightarrow (1-\rho)\tau_{ij} + \Delta\tau_{ij},$$

where

$$\Delta\tau_{ij} = \begin{cases} \rho \cdot \tau_{ij} + 1/C^{best}, & \text{if (i, j) } \in \text{ the tour of the ant that allowed to deposit pherome,} \\ 0, & \text{otherwise,} \end{cases}$$

and $C^{best}$ is the length of tour constructed by the ant that is allowed to deposit pheromone. This ant is either the best-so-far one or the

[15]Its value is either 0 or 1.

iteration-best one or the restart-best one. In general, in MMAS implementations, these three types of ants are used to update in an alternative way.

*Initialize and Restart:* MMAS sets initial pheromone values to an estimate of the upper pheromone value bound. Moreover, it restarts the search progress when the algorithm reaches the stagnation status. This technique is used the same as done in BWAS.

*Introducing limits of pheromone values:* Maximum and minimum values of trail are explicitly introduced. MMAS does not allow trail strengths to get zero value, nor too high value in order to avoid search stagnation. It was shown that, in the long run, the upper pheromone value bound on any arc is limited by $1/pC^{\star}$, where $C^{\star}$ is the length of the optimal tour. Thus, MMAS uses an estimates of this value, $1/pC^{bs}$, where $C^{bs}$ is the length of the best-so-far tour, to define $\tau_{max}$. The lower pheromone value is set to $\tau_{min} = \tau_{max}/a$, where $a$ is a parameter.

## 3.3.2 Issues in Governing the Dynamical Updating in the Trade-Off Technique

### 3.3.2.1 The Updating Function

The dynamical updating rule to $q_0$ is governed by the following linear equation:

$$q_0(t+1) = q_0(t=0) + \frac{(\xi - q_0(t)) \cdot \text{number of current tours}}{\theta \cdot \text{maximum number of generated tours}} \tag{3.3.6}$$

where $t$ is the index of the current iteration, $q_0(t)$ is the value of $q_0$ at the $t$-th iteration, parameters $\xi$ and $\theta$ are used to control the value range of $q_0$ to ensure that the value of $q_0$ is always in a given interval. $\xi$ is set to a smaller value than $q_0(0)$ such that, given $\theta$, we have

$$\frac{\xi \cdot \text{number of current tours}}{\theta \cdot \text{maximum number of generated tours}} \ll q_0(0). \tag{3.3.7}$$

With $(\xi, \theta)$ chosen as in ( 3.3.7 ), it is approximately to have $q_0(t) < q_0(0)$ or hence, from ( 3.3.6 )

$$q_0(0) > q_0(t) > q_0(0) \cdot (1 - \frac{1}{\theta}).$$

So, by selecting suitable values for $(\xi, \theta)$, we can assure that $q_0$ receives only values in any given value interval.

The next section will represent an numerical analysis of adding the pseudo-random proportional rule (with $q_0$ being dynamically adapted according to ( 3.3.6 )) into Ant-based algorithms including MMAS, ACS, and BWAS.

### 3.3.3 Experimental Settings and Analysis of Results

Dynamically updating value of $q_0$ according to ( 3.3.6 ) is carried out either right after all ants finish building their complete tours or at a certain step which they have not finished building those tours yet. To do the later, ( 3.3.6 ) must have a little bit modification. For the sake of simplicity, the former is selected.

Because Ant-based algorithms work better when local search are utilized, we will consider the influence of this new rule in two cases: using local searchs or not. For TSP, a well-known local search named *2-opt* is then selected. The other well-known one is the *3-opt* but this local requests a more

complex implementation and costs much more runtime than *2-opt* does. Because of this reason, we select *2-opt* for our testing purpose. All tests were carried out on a Pentium IV 1.6Ghz with 512MB RAM on Linux Redhat 8.0 platform[16].

#### 3.3.3.1 Without local search

To understand how the pseudo-random transition rule impacts Ant-based algorithms' performance when the value of exploiting parameter is changed, we need to experiment on MMAS, ACS, and BWAS. We first experiment on MMAS and ACS to monitor their performance under the change; if the outcome is positive then we will continue measuring BWAS's performance. As shown later, the outcome is negative thus we only report results on MMAS and ACS. The MMAS and ACS algorithms with the new state transition rule (dynamical updating one) are called MMAS-BNL (MMAS-Balance with No Local search) and ACS-BNL correspondingly.

**MMAS:**

In all tests performed by MMAS-BNL, parameters are set as follows: the number of ants $m = n$ with $n$ being the size of instances, the number of iterations = 10,000. The average solutions are computed after 25 independent runs. Computational results of MMAS-BNL and MMAS are shown in Table (3.1). Here, results of MMAS (without using the trade-

[16]The program we modified to fit our testing purpose is ACOTSP v.1.0 by Thomas Stützle. Source codes of this program can be downloaded at http://iridia.ulb.ac.be/ mdorigo/ACO/aco-code/public-software.html

Table 3.1: Computational results of MMAS and MMAS-BNL. There are 25 runs done, and no local search is used in both algorithms. For MMAS-BNL, $\xi = 0.1$, $\theta = 3$, and $q_0(0) = 0.9$. The number attached with a problem name implies the number of cities of that problem. The best results are bolded.

| | MMAS | | | MMAS-BNL | | |
|---|---|---|---|---|---|---|
| Problem | Best | Avg | $\sigma$ | Best | Avg | $\sigma$ |
| Eil51 | **426 (0.00%)** | **426.7 (0.16 %)** | **0.73** | **426 (0.00%)** | 427.87 (0.44%) | 2.0 |
| KroA100 | **21282(0.00%)** | **21302.80(0.1%)** | **13.69** | **21282(0.00%)** | 21321.72(0.19%) | 45.87 |
| D198 | **15963(1.14%)** | **16048.60(1.70%)** | **79.72** | 15994(1.36%) | 16085.56(1.93%) | 50.37 |
| Att532 | **28000(1.13%)** | **28194.80(1.83%)** | **144.11** | 28027 (1.23%) | 28234.80 (1.98) | 186.30 |

off technique) are quoted from [190]. In order to gain a comparison which is as fair as possible, the parameters setting of MMAS-BNL is the same as that of MMAS in [190]. Values in parentheses in this Table are the relative errors between current values (best and average ones) and the optimal solutions. This error is computed as 100%*(current value - optimal value)/optimal value. [h] From Table (3.1), it shows that performance of MMAS-BNL is worse than that of MMAS. There is no solution quality improvement for any testing instances obtained when the trade-off technique is introduced if not using local search.

**ACS**

We carry out experiments for ACS-BNL with parameter settings which are the same as in [80]. The settings are as follows: the number of ants $m = 10$, $\beta = 2.0$, $\rho = \alpha = 0.1$. The number of iterations is computed as $it = 100 * problem\ size$, hence the number of generated tours will be $100 * m * problem\ size$, where *problem size* is the number of cities. Except the result of ACS for *pcb*442 instance obtained from our implementation, results in Table (3.2) of ACS on selected testing instances of

Table 3.2: Computational results of ACS and ACS-BNL. There are 15 runs done, and no local search is used in both algorithms. For MMAS-BNL, $\xi$ = 0.1, $\theta$ = 3, and $q_0(0)$ = 0.9. The number attached with a problem name implies the number of cities of that problem. The best results are bolded.

| | ACS | | | ACS-BNL | | |
|---|---|---|---|---|---|---|
| Problem | Best | Avg | $\sigma$ | Best | Avg | $\sigma$ |
| Eil51 | **426 (0.00%)** | **428.06 (0.48 %)** | **2.48** | **426 (0.00%)** | 428.60 (0.61 %) | 3.45 |
| KroA100 | **21282(0.00%)** | **21420(0.65%)** | **141.72** | **21282(0.00%)** | 21437(0.73%) | 234.19 |
| Pcb442* | **50778(0.00%)** | **50778(0.00%)** | **0.0** | **50778(0.00%)** | 50804.80(0.05%) | 55.48 |
| Rat783 | **9015(2.37%)** | **9066.80(2.97%)** | **28.25** | 9178(4.22%) | 9289.20(5.49%) | 70.16 |

TSP is recalled from [80]. Values in parentheses in this Table are the relative errors between current values (best and average ones) and the optimal solutions. This error is computed as 100%*(current value - optimal value)/optimal value. Numerical results for ACS and ACS-BNL are shown in Table (3.2). In comparison with results of ACS which are cited from [80], we see that ACS-BNL found the best solutions for small scale instances like *eil51, KroA100, Pcb442* and so did ACS. But the average solutions and values of the standard deviation found by ACS for those instances are better than that by ACS-BNL. Moreover, ACS performs better than ACS-BNL does on *rat783* a large instance in terms of measures of best solution, average solution, and standard deviation. Without using local search, ACS outperforms ACS-BNL in all test instances.

#### 3.3.3.2 With local search

MMAS, BWAS, and ACS are the Ant-based algorithms chosen for this investigation purpose. The modified versions of MMAS, BWAS, and ACS algorithms with the new state transition rule are called MMAS-BL (MMAS-Balance with Local search), BWAS-BL, ACS-BL respectively. Results of the original MMAS were taken from [190] while that of the original BWAS and ACS were from [44] and [80] respectively. Values in parentheses in this Table (3.3) are the relative errors between current values (best and average ones) and the optimal solutions. This error is computed as 100%*(current value - optimal value)/optimal value.

**MMAS:**

In [190], Stützle studied the importance of adding local search into MMAS with the consideration that either all ants perform a local search or only the best one does so. In addition, in his study, the number of ants is also considered. Thus, there are three versions of MMAS with local search added including: 10 ants used and all ants do local search (named *10+all-ls*), 10 ants used and only the best ant does local search (*10+best-ls*, and the last version which the number of ants used is equal to the number of cities of TSP instance and only the best ant performs local search (named *MMAS+ls*). We mentioned here *10+all-ls* and *MMAS+ls* versions since it was claimed that in long run these two are better than the rest (*10+best-ls*). To make the comparison fairly, all systematic parameters of MMAS-BL were set equally to that of *10+all-ls*. Settings are: number of ants $m = 10$, number of nearest neighbor = 35, evapo-

Table 3.3: MMAS variants with 2-opt for symmetric TSP. The runs of MMAS-BL were stopped after $n \cdot 100$ iterations. The average solutions were computed for 10 trials. In MMAS-BL, $m = 10$, $q_0(0) = 0.9$, $\rho = 0.99$, $\xi = 0.1$, and $\theta = 3$. The best results are bolded. The number attached with a problem name implies the number of cities of that problem. The best results are bolded.

| | | MMAS: $n \cdot 100$ iterations | | MMAS: $n \cdot 2500$ iterations | |
|---|---|---|---|---|---|
| Problem | MMAS-BL | 10+all-ls | MMAS-ls | 10+all-ls | MMAS-ls |
| KroA100 | **21282.00(0.00%)** | 21502(1.03%) | 21481(0.94%) | **21282(0.00%)** | **21282(0.00%)** |
| D198 | **15796.20(0.10%)** | 16197(2.64%) | 16056(1.75%) | 15821(0.26%) | 15786(0.04%) |
| Lin318 | **42067.30(0.09%)** | 43677(3.92%) | 42934(2.15%) | 42070(0.09%) | 42195(0.39%) |
| Pcb442 | **50928.90(0.29%)** | 53993(6.33%) | 52357(3.11%) | 51131(0.69%) | 51212(0.85%) |
| Att532 | **27730.50(0.16%)** | 29235(5.59%) | 28571(3.20%) | 27871(0.67%) | 27911(0.81%) |
| Rat783 | **8886.80 (0.92%)** | 9576 (8.74%) | 9171 (4.14%) | 9047 (2.74%) | 8976 (1.93%) |

ration factor $\rho = 0.99$, $\alpha = 1.0$, $\beta = 2.0$, all ants are allowed to perform local search. It is noteworthy that the maximum number of iterations of MMAS-BL for an instance of size $n$ is $n \cdot 100$ which implies that the number of generated tours of MMAS-BL is $m \cdot n \cdot 100$. [h]

Comparing performance of MMAS-BL with performance of both $MMAS - ls$ and $10 + all - ls$ can be shown in Table (3.3). For the problem *rat*783, even though only with 5000 iterations performed, MMAS-BL still outperformed the other two algorithms (much more number of iterations given to those two algorithms). In all tests, both small and large scale instances, performance of MMAS-BL is always over *MMAS-ls* and *10+all-ls* even though the number of generated tours of MMAS-BL is much less than or equal to that of the other two.

**BWAS:**

Parameters setting for experiments for BWAS with the dynamically updating trade-off technique (BWAS-BL) is the same that for BWAS in [44].

Table 3.4: Parameter and configuration of the local search procedure in BWAS

| Parameter | Value |
|---|---|
| No. of ants | $m = 25$ |
| Maximum no. of iterations | $Nit = 300$ |
| No. of runs | 15 |
| Pheromone updating rules parameter | $\rho = 0.2$ |
| Transition rule parameters | $\alpha = 1$, $\beta = 2$ |
| Candidate list size | $cl = 20$ |
| Pheromone matrix mutation prob. | $P_m = 0.3$ |
| Mutation operator parameter | $\sigma = 4$ |
| % of different edges in the restart condition | 5% |
| No. of neighbors generated per iteration | 40 |
| Neighbor choice rule | *1st improvement* |
| *Don't look bit structure* | used |

Let us recall the table of parameters values of BWAS in [44] described in Table (3.4). Results of BWAS and BWAS-BL are represented in Table (3.5).

Except for *Berlin51* to which the performance of BWAS and that of BWAS-BL are the same, from Table (3.5), it has been seen that despite obtaining the optimal solution, the average solution of BWAS-BL is marginally worse than that of BWAS on small scale instances like *Eil51, KroA100*. In contrast, on larger scale instances, like *att532, rat783, fl1577*, BWAS-BL performs significantly better than BWAS in terms of measures of best-found solution, average solution, and standard deviation. Except the instance *fl1577* where standard deviation of BWAS-BL is worse than

Table 3.5: Compare performance between the BWAS algorithm with its variant utilizing the trade-off technique. In BWAS-BL, $\xi = 0.1$, $\theta = 3$, and $q_0(0) = 0.9$. The optimal value of the corresponding instance is given in the parenthesis. The best results are bolded. $Error = \frac{bestvalue - optimalvalue}{optimalvalue} * 100\%$.

| | **Eil51 (426)** | | | | | **Att532 (27686)** | | | |
|---|---|---|---|---|---|---|---|---|---|
| Model | Best | Average | Dev. | Error | Model | Best | Average | Dev. | Error |
| BWAS | **426** | **426** | **0** | **0** | BWAS | 27842 | 27988.87 | 100.82 | 1.09 |
| BWAS-BL | **426** | 426.47 | 0.52 | 0.11 | BWAS-BL | **27731** | **27863.20** | **84.30** | **0.64** |
| | **Berlin52 (7542)** | | | | | **Rat783 (8806)** | | | |
| Model | Best | Average | Dev. | Error | Model | Best | Average | Dev. | Error |
| BWAS | **7542** | **7542** | **0** | **0** | BWAS | 8972 | 9026.27 | 35.26 | 2.50 |
| BWAS-BL | **7542** | **7542** | **0** | **0** | BWAS-BL | **8887** | **8922.33** | **16.83** | **1.32** |
| | **KroA100 (21282)** | | | | | **Fl1577 (22249)** | | | |
| Model | Best | Average | Dev. | Error | Model | Best | Average | Dev. | Error |
| BWAS | **21282** | **21285.07** | **8.09** | **0.01** | BWAS | 22957 | 23334.53 | **187.33** | 4.88 |
| BWAS-BL | **21282** | 21286.60 | 9.52 | 0.02 | BWAS-BL | **22680** | **23051** | 351.87 | **3.60** |

Table 3.6: Parameter and configuration of the local search procedure in ACS

| **Parameter** | **Value** |
|---|---|
| No. of ants | $m = 10$ |
| Maximum no. of iterations | *Nit = 100* problem size* |
| No. of runs | 15 |
| Pheromone updating rules parameter | $\rho = \alpha = 0.1$ |
| Transition rule parameters | $\beta = 2$ |
| Exploiting parameter | $q_0 = 0.98$ |
| Candidate list size | *cl = 20* |

that of BWAS, for other instances the inversion is held.

**ACS:**

Parameters setting for experiments for ACS which is incorporated with the dynamically updating trade-off technique - call *ACS-BL* - is the same

Table 3.7: Compare performance between the ACS algorithm with its variant ACS-BL utilizing the trade-off technique. In ACS and ACS-BL, $\xi = 0.1$, $\theta = 3$, and $q_0(0) = 0.98$. The optimal value of the corresponding instance is given in the parenthesis. The best results are bolded. $Error = \frac{bestvalue - optimalvalue}{optimalvalue} * 100\%$

| | **D198 (15780)** | | | | | **Pcb442 (50779)** | | | |
|---|---|---|---|---|---|---|---|---|---|
| Model | Best | Average | Dev. | Error | Model | Best | Average | Dev. | Error |
| ACS | 15888 | 16054 | 71 | 0.68% | ACS | 51268 | 51690 | 188 | 0.96% |
| ACS-BL | **15,780** | **15,880.67** | **52** | **0.0%** | ACS-BL | **50779** | **50872.33** | **156.3** | **0.0** |
| | **Att532 (27686)** | | | | | **Rat783 (8806)** | | | |
| Model | Best | Average | Dev. | Error | Model | Best | Average | Dev. | Error |
| ACS | 28147 | 28523 | **275** | 1.67% | ACS | 9015 | 9,066 | 28 | 2.37% |
| ACS-BL | **27752** | **27933.56** | 289 | **0.24%** | ACS-BL | **8902** | **9012.33** | **16.83** | **1.01%** |
| | **Fl1577 (22249)** | | | | | | | | |
| Model | Best | Average | Dev. | Error | | | | | |
| ACS | 22977 | 23,163 | **116** | 3.27% | | | | | |
| ACS-BL | **22680** | **23082** | 204 | **1.94%** | | | | | |

that for ACS in [80]. Parameter values of ACS in [80] are described in Table (3.6). Results of ACS and ACS-BL are represented in Table (3.7). Take note that, ACS was using a 3-opt procedure in its local search procedure. 3-opt usually produces better solutions than 2-opt as 3-opt's search space covers 2-opt's. This local search is originated from a k-opt local search [17]. From Table (3.7), it has been seen that except for instances *Att532, Fl1577* only on which ACS-BL is outperformed by ACS on the standard deviation metric only, on remaining instances and remaining metrics ACS-BL is significantly superior than ACS.

#### 3.3.3.3 Discussion

As shown in the above computational results, the trade-off technique or pseudo-random proportional rule with a dynamical updating technique is an efficient

[17] An illustration of k-opt can be found in the definition 2.2.2 on page 52.

and effective tool in improving solution quality of MMAS, BWAS, and ACS when there is the presence of local search in these algorithms. Indeed, results from Table (3.3) showed that MMAS-BL presents a better performance than MMAS. It outperformed its testing counterpart for all six test instances within smaller number of iterations. Also, from tables (3.5, 3.7), BWAS-BL and ACS-BL proved the effectiveness and usefulness of this modified trade-off technique by outperforming original BWAS and ACS respectively for large instances.

However, without using local search procedures, Ant-based algorithms incorporating this technique performed worse than that which are not using this technique. This claim is supported by obtained numerical results. But, it is worth mentioning here that most experimental works in literature revealed that performance of Ant-based algorithms is much improved if local search procedures are utilized. Thus, the solution quality improvement of this trade-off technique with presence of local search is more impressive and worthily attentive; hence its failure to improve solution quality when local search procedure is absent can be tolerable. Possible explanations of why the proposed technique is able to provide better quality solutions only for algorithms using local search procedures are of following:

- A local search aims at obtaining better solutions in the range of current best solutions which might lead to early stagnation of algorithms. The proposed technique aims at avoiding that stagnation by giving more chance (higher probability) to search through unvisited regions. This technique is designed to be more effective at the early and late stages of search process. If the value of exploiting parameter is fixed, let's say to

a high value, and then the search is focusing heavily on those first few good solutions found in the early stage of the search and less chance of guiding to visit to regions that are not traveled, i.e. less chance of improving quality at the late stage. Similar explanation to if the parameter is set to a low or medium value or in other words to a fixed value.

- The proposed technique has the role of introducing unvisited region to the search process and the local search has the role of extensively searching through the regions to get very good solutions. Therefore, more regions containing local optimum solutions will be searched through. If using the proposed technique without using local search, the search process will similarly travel to more unvisited regions but will not search extensively enough in those regions hence good solutions in those regions receive less chance to be visited.

## 3.4 Chapter Summary

We investigated the convergence of a more general version of GBAS in section 3.2. From the experimental aspect, our convergence results has drawbacks of designing implementations in terms of this generalized framework as that seen in GBAS. Indeed, a high convergent probability can be gained by setting a very large number of agents and/or a small value of evaporation parameter. And obtaining those results may cost a hardly acceptable amount of computation time if run on a single-processor machine. Moreover, there is a similarity between our convergent results and Gutjahr's about the restriction to the application range because of the appearance of the uniqueness

conditions (which are removed in [108]) in both frameworks.

However, our results are different from theoretical results in the previous works in some aspects. First of all, our results seem to be closer to implementations; one such example is the ACS algorithm from which its trade-off mechanism is adopted and analyzed in our framework . In actual implementations, the exploiting parameter $q_0$ is usually set very close to 1 which means that the probability of choosing the next node according to ( 3.2.1 ) is very small (equal to $1-q_0$), hence GBAS is less fitting to explanation of behavior of implementations than EGBAS. Secondly, the convergence speed of our model might be controlled not only by systematic parameters as in GBAS but also by the exploiting parameter. This seems that implementations based on our model can have better flexibility in controlling the search progress than those based on GBAS.

In section 3.3, a variant of pseudo-random proportional transition rule with linearly dynamical updating techniques is proposed and empirically analyzed. Experimental analysis of incorporating this rule into some state-of-the-art Ant-based algorithms is carried out for Traveling Salesman Problem. Computational results showed that this dynamical updating rule enhances some Ant-based algorithms' performance if local search procedures are in use.

# Chapter 4

# Decomposition-based Search Approach

## 4.1 Background and Introduction

In the previous chapter (chapter 3), we have presented our findings related to ACO which is nature-inspired metaheuristic to solve a broad class of combinatorial optimization problems. ACO has shown a good performance on optimization problems of small and medium size. For those of large size, for example instances of Traveling Salesman Problem (TSP), ACO's running time is increased due to its algorithmic complexity which is at least $O(n^2 * m)$ without using local search and $O(n^3 * m)$ with local search embedded (for example, see [65])[1]. New approaches to solve large size instances more effectively are therefore needed. In this chapter, we will mainly present results of analysis of a method based on a recently emerged approach that is to combine a metaheuristic with classical decomposition techniques in Artificial Intelligence (AI) and Operation Research (OR) to tackle large size instances of optimization problems. The method going to be presented is a combination be-

[1] $m, n$ are the number of artificial ants and the number of cities of TSP respectively. It is not mathematically rigorous to define "the algorithmic complexity of a metaheuristic" due to variety of algorithms (for certain problems) built around the metaheuristic. Each of these algorithms has its own algorithmic complexity. However, in this case, we refer the algorithmic complexity of ACO (for TSP) in terms of the algorithmic complexity of ACO's state-of-the-art algorithms (for TSP) which include ACS and MMAS.

tween decomposition (or clustering) techniques with an available problem-solving method, especially a metaheuristic [2]. Hence, the method based on the combination is henceforth called *decomposition-based* method. In contrast to nature-inspired metaheuristics, for example GA and ACO, whose problem-solving capability is from modeling certain features of nature, the idea of decomposition-based methods is to make full use of problem domain knowledge, especially the mathematical structures of optimization problems, to effectively tackle large size instances by simply solving a number of smaller-in-size instances of those problems.

Although the idea of decomposing a problem to simplify its solution is not new, the use of large scale computers in recent years has led to rapid expansion of decomposition techniques for optimization, for solving reliability and electrical network problems, for process control, and in a wide variety of other problems.

The fact is that physical systems are growing bigger and bigger in size. The size of inherent optimization problems associated with these systems whose performance and efficiency are always required to be improved is therefore growing by time. Thus, there has been increasing demand of solving large scale instances[3] of optimization problems and that demand is from both academic and industrial interests. Some nature-inspired metaheuristics

[2]The fact is that the analysis for decomposition-based method in this chapter can be interpreted straightaway to any other methods to solve COPs.

[3]It is worthy of notice that it is subjective to judge whether an instance of a certain optimization problem is "large scale" or not. A sound judgement should take into account impacting factors like the hardness of the problem in terms of complexity theory, the performance of state-of-the-art algorithms

have shown the drop of their performance when applied to large scale instances of optimization problems for which they have achieved a good performance when applied to small or moderate scale instances. One of reasons that led to their decreasing performance is the high complexity of their algorithms. It is beneficial to use existing algorithms to solve large scale problem instances through solving smaller scale ones if we can utilize the mathematical structures of the problem to "break" a large scale instance into small or medium scale parts that are efficiently and effectively solved by existing algorithms and then assemble solutions found for those parts to form a solution to the large scale instance.

In one portion of this chapter, we will present our findings of what we benefit from these methods in terms of empirical runtime. The idea of decomposition-based approach is based on the well-known principle called "Divide and Conquer".

**Divide and Conquer Principle** Divide and Conquer (derived from the Latin saying *Divide et impera* and called DC for short) is an important algorithm design principle. The philosophy of the principle is to solve many simple problems rather than solve a difficult problem. It works by recursively breaking down a problem into two or more *sub-problems* of the same (or related) type, until these become simple enough to be solved directly by a specific problem-

for the problem, and the strength of the present computational technology. However, it is sufficient to serve our purpose of this study when we use a generic concept of "large scale"; thus if we henceforth refer to a large scale instance of an optimization problem we mean it a generic large scale instance.

solving algorithm[4]. The solutions to the subproblems[5] are then combined to give a solution to the original problem.

Many well-known, effective and efficient algorithms based on DC were proposed and developed such as sorting (e.g. quicksort, mergesort), searching (e.g. binary search, depth-first search) and even the fast Fourier transform algorithm (refer to [2, 45] for more details about DC's applications). There have also been numerous algorithms built around this principle and aimed at solving large scale optimization problems in literature.

Based on the guarantee of the optimality of solutions of those algorithms, we classified them into two types: exact and heuristic decomposition-based algorithms. Brief reviews of both types will be presented in the next two subsections.

### 4.1.1 Overview of chapter

A typical *exact* decomposition-based algorithm, that is well-known in the mathematical programming area, by Dantzig and Wolfe [50] - now commonly referred to as Dantzig-Wolfe Decomposition Principle - is one of those reviewed algorithms. Having similar characteristics to exact methods, exact decomposition-based approaches have been designed to aim at obtaining optimal solutions of underlying optimization problems. In contrast, the

[4]The algorithm used in this stage, which is so-called "the conquer stage" in literature, to find solutions to subproblems is called the conquering-stage algorithm in the rest of this paper.

[5]Hereafter, if there is not any confusion then the word "subsolution" will be used for the phrase "solution of sub-problem" with the same meaning.

heuristic decomposition-based have been designed to compromise between runtime and optimality of solutions. The inherent disadvantage of exact approaches is still the runtime. As presented in section 4.1.2, Dantzig-Wolfe Decomposition Principle cannot predict how long the algorithm needs to run before achieving the optimal solution. However, in general decomposition-based methods can reduce their runtime by using parallel computation in their iterative subproblems-solving process.

An advantage of decomposition-based methods is that their implementations run fast if their decomposition and recombination procedures are well-designed [6, 178, 192]. Assume these procedures are designed well enough, our study focuses on how fast they are in comparison to non-decomposition-based methods. Our first findings which are presented in the section 4.2 show that the decomposition-based methods are faster than their non-decomposition-based competitors but cannot be faster than a certain times which is computed through both the algorithmic complexity of the procedures to solve subproblems and the number of resulted subproblems. As accounted for later, another advantage of decomposition-based methods is the positive effect on memory requirement.

In this part of study, we focus more on inexact decomposition-based approaches. Parallel to the development of exact decomposition-based methods, the start of inexact decomposition-based methods can be traced back in '60s [6]. Many applications in optimization field using inexact decomposition-based methods can be found in literature ( [121, 170, 178, 192, 195]). Refer back in subsection 4.1.2.1 (on page 133) for the review on POPMUSIC method which is regarded as an inexact decomposition-based method.

We also propose and test a simple heuristic decomposition-based method for Euclidean Traveling Salesman Problem (see section 4.3). In addition to the numerical simulation results, we show a sufficient condition for the method which guarantees that the optimal solutions are included in search space while the size of the search space is reduced. The result may appear trivial due to its less impacting to other works, however, the motivation behind the finding is very important to the field if we look at a disadvantage existing in most of heuristic decomposition-based methods in literature. Their disadvantage of not able to guarantee the optimality of solutions is actually the main challenging in their applications. Lack of a sufficient condition (for certain methods applying to certain problems) to guarantee that the optimal solutions belong to search space is one reason among reasons causing the disadvantage.

The structure of this chapter is organized as follows. Exact decomposition-based methods which originate from the principle of divide and conquer will be reviewed in subsections 4.1.2, while review of inexact decomposition-based method can be referred to subsection 4.1.2.1 After this introduction section,the next section will be devoted to our theoretical findings which are related to relative upper and lower bounds of runtime of decomposition-based methods. We will account for how these findings fit into the context of some decomposition-based methods in literature in subsequent subsections. Also in that section, we illustrate the usage of decomposition-based into two problems, each for a different illustrative purpose. For the first problem which is named 2D protein folding problem, we illustrate how decomposition-based approach helps to design a new way of representing protein conformations

on 2D grid to reduce search space for Genetic Algorithm. For the second one named Euclidean Traveling Salesman problem (ETSP), in section 4.3 we prove a sufficient condition on the geometrical structure of ETSP instances for a simple heuristic decomposition-based algorithm. The condition guarantees that all optimal solutions to original instance must be embraced in the search space of that algorithm. What we did to prove that condition is actually addressing a concern over decomposition-based methods about whether or not those methods converge to optimal solutions [6]. Also for ETSP, we presented numerical results for a decomposition-based method that using ACS to solve ETSP in section 4.4. Some tested instances are from benchmark library while the others are generated using a random generator of ETSP instances. Finally, a summary will end the chapter.

### 4.1.2 Dantzig-Wolfe Decomposition Principle in Mathematical Program

It often happens that a large linear program is really a collection of smaller linear programs that are largely independent of each other. As an example, suppose we have some LP where constraint matrix is the node-arc incidence matrix of a very large graph, but the graph has the form shown in Fig.4.1.

Arcs exist only between nodes of the same sets and between the set $N_C$ and the others. The Dantzig-Wolfe decomposition method is designed to take

[6]Given an optimization problem and a decomposition-based method to solve it, if we can show that the search space of the method does not include any optimal solution, then indirectly we do reject any claim for the convergence of the method.

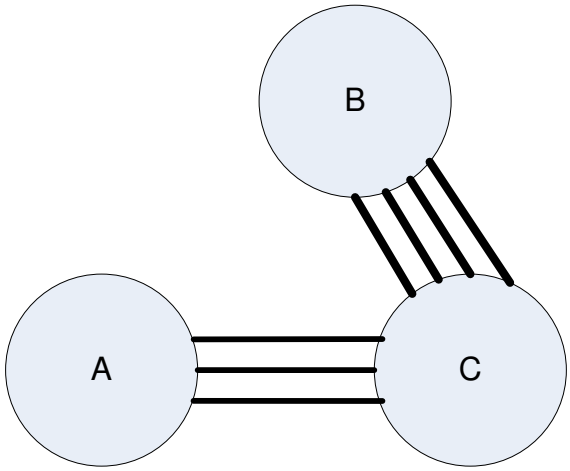


Figure 4.1: A graph whose node-arc incidence matrix is decomposable.

advantage of this special structure by allowing us to solve the entire problem by iteratively solving subproblems with sizes of A and B. Using the above graph to represent node-arc relationship for a LP whose constraint matrix is formulated as follows:

$$\left[\begin{array}{c|c} \begin{matrix} n_1\text{cols.} \\ D \end{matrix} & \begin{matrix} n_2\text{cols.} \\ F \end{matrix} \\ \hline A & 0 \\ \hline 0 & B \end{array}\right] \begin{matrix} \\ \}\, m_0\text{rows} \\ \}\, m_1\text{rows} \\ \}\, m_2\text{rows} \end{matrix} \tag{4.1.1}$$

with variables $x \in \mathbb{R}^{n_1}$ corresponding to the first $n_1$ columns and $y \in \mathbb{R}^{n_2}$ corresponding to the next $n_2$ columns. The complete LP can be written in

standard form as

$$\begin{aligned} \min z = \quad & c'x + d'y \\ & Dx + Fy \quad = b_0 \\ & Ax \qquad = b_1 \\ & By \qquad = b_2 \\ & x, y \qquad \geq 0 \end{aligned} \tag{4.1.2}$$

The first $m_0$ equation is called the *coupling equations*; the problems associated with the succeeding sets of rows are called *subproblems* A and B, respectively. In particular, constraints of the subproblem A are considered as

$$\begin{aligned} Ax \quad &= b_1 \\ x \quad &\geq 0 \end{aligned} \tag{4.1.3}$$

Since any feasible point in this subproblem can be written as a convex combination of vertices of the feasible set. Call these vertices $x_1, ..., x_p$, and write

$$\begin{aligned} x \qquad &= \sum_{j=1}^{p} \lambda_j x_j \\ \lambda_j \qquad &\geq 0 \\ \sum_{j=1}^{p} \lambda_j \quad &= 1 \end{aligned} \tag{4.1.4}$$

Similarly, we write

$$\begin{aligned} y \qquad &= \sum_{j=1}^{q} \mu_j y_j \\ \mu_j \qquad &\geq 0 \\ \sum_{j=1}^{q} \mu_j \quad &= 1 \end{aligned} \tag{4.1.5}$$

where the $y_j$ are the vertices of subproblem B.

Replacing $x$ and $y$ by their representations in ( 4.1.4 ) and ( 4.1.5 ), ( 4.1.2 ) becomes the LP in the variables $\lambda_j$ and $\mu_j$ shown below

$$\begin{array}{lcccc} \text{variables} & : & \lambda_1 ... \lambda_p & \mu_1 ... \mu_q & \\ m_0 \text{rows} & \{ & \Delta_1 ... \Delta_p & \Phi_1 ... \Phi_q & b_0 \\ 1\text{rows} & \{ & 1...1 & 0...0 & 1 \\ 1\text{rows} & \{ & 0...0 & 1...1 & 1 \end{array} \tag{4.1.6}$$

where

$$\begin{aligned} \Delta_j &= Dx_j, \quad j = 1, ..., p \\ \Phi_j &= Fy_j, \quad j = 1, ..., q \end{aligned} \tag{4.1.7}$$

and the cost is

$$\min z = \theta' \lambda + \sigma' \mu \tag{4.1.8}$$

where

$$\begin{aligned} \theta' &= c' x_j, \quad j = 1, ..., p \\ \sigma' &= d' y_j, \quad j = 1, ..., q \end{aligned} \tag{4.1.9}$$

and

$$\lambda, \mu \geq 0 \tag{4.1.10}$$

are the new variables, in $\mathbb{R}^p$ and $\mathbb{R}^q$, respectively. The problem in this form is usually called the *master problem*.

We may describe the operation of the decomposition method, that relies on a so-called *Revised Simplex Method* [15, 130, 154], in the following terms. The master problem, based on its overall view of the entire situation, sends a price to subproblem A. This subproblem then responds with a solution (called a proposal) for possibly improving the overall problem, based on its local information and the price. The master problem then weighs the cost of this proposal against its criterion $\alpha$ for subproblem A. If the proposal is cheaper than $\alpha$, it is implemented by bringing it into the basis. If not,

subproblem B is sent a price and asked for a proposal. As long as a subproblem can produce a favorable proposal, the master problem can find a favorable pivot. When neither subproblem can come up with a favorable proposal, we have reached an optimal solution for the entire problem [50]. This decomposition-based approach and its extensions are actually favored to solve large scale instances than to solve small or moderate ones.

#### 4.1.2.1 Partial Optimization Metaheuristic Under Special Intensification Conditions

POPMUSIC which is considered as a metaheuristic, stands for Partial Optimization Metaheuristic Under Special Intensification Conditions [196]. The idea of this metaheuristic is to locally optimize sub-parts of a solution rather than globally optimize the whole solution. Those sub-parts are found by a certain clustering algorithm (one example is *k-mean clustering* [70]). There are number of different approaches sharing the similar idea with POPMUSIC principles and mentioned in [196] as well. The algorithmic framework (16) below restates the general framework of POPMUSIC in [196].

To use POPMUSIC, a solution to the optimization problem in question is assumed to be able to be presented as a set of $p$ parts $s_1, \ldots, s_p$. Some parts are more in relation with other parts, and furthermore, it is posited that one is able to define the relatedness measure between two parts. POPMUSIC first chooses a part so-called *seed* $s_i$ and a number $r < p$ of other parts which are most related to $s_i$. A sub-problem $R_i$ is then formed by those most related parts. Both of these procedures (disassembling into parts and assembling parts into a smaller-in-size problem) are carried out at step 4 and 5 of

**Algorithm 16** POPMUSIC metaheuristic

1: Input: Solution $S$ composed of parts $s_1, \ldots, s_p$
2: Set $O = \emptyset$
3: **while** $O \neq \{s_1, \ldots, s_p\}$ **do**
4: Select $s_i \notin O$
5: Create a sub-problem $R_i$ composed of the $r < p$ parts $s_{i_1}, \ldots, s_{i_r}$ most related to $s_i$
6: Optimize $R_i$
7: **if** $R_i$ has been improved **then**
8: Update $S$ (and corresponding parts) and set $O \leftarrow \emptyset$
9: **else**
10: Set $O \leftarrow O \cup \{s_i\}$
11: **end if**
12: **end while**

Alg. (16). $R_i$ is next optimized at the step 6. Thus, rather than applying an optimizer to the entire of a given solution, POPMUSIC does *locally* which means seeking optimal solutions to the sub-problem $R_i$ only. Moreover, in [196], by experimental results they claimed that POPMUSIC ran faster than the procedure of optimizing the entire of the original solution (roughly speaking, this procedure is likely POPMUSIC at $p = 1$).

#### 4.1.2.2 Decomposition-based method's advantage of less storage space requirement

The effect on space requirement is an obvious advantage of the decomposition algorithm[7]. In any reasonable example, we now have a large number of columns, one for each vertex of each of the two subproblems. But the number of rows has been reduced from $m_0 + m_1 + m_2$ to $m_0 + 2$ (see (4.1.1) and (4.1.6))

[7]The effect on space requirement by the decomposition method presented for Linear Programming problems in this portion is actually similar to that by our heuristic decomposition-based method introduced in later section.

and Revised Simplex method can be implemented with a "working" matrix of size $(m_0 + 3) * (m_0 + 3)$. Putting all other considerations aside, this means that we can fit much larger problems into a fast-access storage. We now require $(m_0 + 3)^2$ storage cells for the working matrix of the master problem, as opposed to $(m_0 + m_1 + m_2)^2$ in the original formulation.

A little arithmetic shows the effectiveness of the approach in fitting large problems into a computer. Suppose we have 100 subproblems each with 1000 rows and 1000 coupling equations ($m_0 = 1000$). Then the original problem has $1000 * 100 + 100 = 100100$ rows; its working matrix will have $100100^2 \approx 10^10$ entries, which cannot fit in the fast memory of any computer known to us. On the other hand, the working matrix of master problem will have $(1000 + 100 + 1)^2 = 1101^2 \approx 10^6$ entries, which is practical to store in the fast memory of any reasonably moderate computer nowadays.

However, it is not clear about its effect on time requirement due to the hardness in estimating how many times the subproblems must be solved to achieve an optimal solution (see Chapter 4 in [160]). As seen later in section 4.2, our results can be used to answer that concern.

## 4.2 Runtime efficiency of decomposition-based methods and their non-decomposition-based counterparts

### 4.2.1 Introduction and Notations

In the previous section we know that decomposition-based methods first decompose a problem instance into several subproblems whose type is the same as or alike to that of the original problem and size is smaller than the origi-

nal instance. A specific algorithm is deployed to solve those subproblems at the second stage that is termed as *conquering-stage* and the algorithm is called *conquering-stage algorithm*. All subproblems can be solved parallel or serially and called parallel decomposition-based or serial decomposition-based approaches, respectively. All solutions to those subproblems are then combined by a certain combining method to form a solution to the original instance. In this part of study, we are going to focus on analyzing the empirical runtime of serial decomposition-based approaches conceptually and compare their runtime-oriented performance to that of approaches using only the specifically problem-solving algorithm (in the conquering-stage). For the sake of simplicity, given a serial decomposition-based method, we name it *SDEB*[8] and abbreviate the counterpart method using its specifically problem-solving algorithm to solve only original instance *PUND* [9] of the corresponding SDEB method[10].

Recursiveness is the most simple and straightforward way of implementing divide and conquer-based methods generally and a SDEB specifically. However, recursive procedures always require long runtime in case of inputs with large size. Therefore, non-recursive and decomposition-based approaches offer promising alternatives for these large size inputs. Published literature suggests that decomposition-based methods using metaheuristics

[8]Serial DEcomposition-Based method.

[9]PUrely Non-Decomposition-based counterpart.

[10]Thus, if we say A is a PUND implementation of the SDEB B (or B's SDEB), then we shorten the statement that the conquering-stage algorithm used to solve subproblem in B is actually used in A to solve solely the same problem type.

as the conquering-stage algorithm can produce satisfactory solutions to real-world instances of NP-hard optimization problems in a short runtime (e.g. Taillard [195], Reimann et. al. [170] solved Vehicle Routing Problem - VRP; Mulder et. al. [149] solved Traveling Salesman Problem - TSP).

There have been experimental achievements in solving NP-hard combinatorial optimization problems using approaches that have very similar features with SDEB approaches. Below are instances of such attempts in literature.

1. Taillard [195] used TS - a well-known metaheuristic ( [42], [90], [96]) - in the conquer stage of solving a very large VRP instances (as large as 199 customers, actually this is a very large input size to practical applications) proposed by Christofides et. al. [29]. In [195], he suggested two methods of decomposing (clustering). Taillard's algorithm (a kind of SDEB) not only ran faster in comparison to those of Gendreau [84] and Osman [155] (the two methods in the later are PUND-alike approaches of Taillard's SDEB), but also produced better solutions to 5 of the 14 test instances and found identical results for the remaining 9 ones.

2. Recently, Reimann et. al. [170] solved such large scale VRP instances with the same approach as Taillard's (SDEB approach) but used different algorithms for the clustering and conquering stage (equivalently, Reimann et. al.'s PUND differs from Taillard's PUND one. In the decomposing stage, the former used the modified Miehle algorithm [142] to determine the *gravity center* of each route, and centers of those routes are then decomposed (clustered) by Sweep algorithms of Gillett and Miller [86]. Finally, the subsolutions are found by a conquering-stage

algorithm so-called Savings-based Ant System (SBAS), which is derived from ACO (readers interested to know more about ACO may refer to Chapter 3). Performance of Reimann et. al.'s SDEB algorithm is similar to that of Taillard's but noticeably it outperforms the original SBAS - the PUND version of Reimann et. al.'s SDEB method - for large instances in terms of both performance measures on runtime and quality of solutions.

3. Mulder et. al. [149] used the a SDEB to solve TSP instances whose size is unimaginably big, up to $1,000,000$ cities. In this algorithm, the decomposing (clustering) ability of Adaptive Resonance Neural Network was used to break TSP instances into subproblems; and the Lin-Kernighan (LK) [135] algorithm - the most successful heuristic particularly designed for TSP - was used as the conquering-stage algorithm. In comparison to some well-known successful algorithms designed also particularly for TSP, such as the original LK [135] (this is the PUND implementation of Mulder et. al.'s SDEB), chained-LK algorithms [4, 136], and Concorde package [11], Mulder's experimental results showed a significant speedup and a good scale of solution quality. In more details, comparing to Concorde package at the $1,000,000-$city level, their algorithm took 16% as long to run and was 11% off of tour quality.

Most of all experimental studies have so far determined the fact that given a big size instance, SDEB implementations always run faster than their PUND counterpart. As a result, it demands theoretical studies that can explain these experimental findings. Why SDEB implementations usually run faster than

[11] Http://www.tsp.gatech.edu/concorde/download.html.

their PUND counterparts? To what the *speedup* of a SDEB implementation's conceptually empirical runtime in relative to its PUND implementation's is related: number of subproblems, the complexity of the conquering-stage algorithm or any thing else? By speedup we mean the ratio of the empirical runtime of a SDEB algorithm to empirical runtime of its PUND one. The speedup concept can also be found in, for examples, Taillard [195] and Mulder [149].

However, no theoretical evidence has been published so far that proves these facts. This article points out such an evidence by stating that the speedup of a SDEB implementation relative to the corresponding PUND counterpart has its own **finite lower and upper bounds** that can be evaluated. This speedup , under some fairly weak conditions, is limited by an upper bound which can be computed from the number of subproblems and *the conceptual empirical runtime function of the conquering-stage algorithm on inputs of a given length* [12]; and by the lower bound of 1. Moreover, it is able to obtain a higher lower bound that is much greater than 1 by changing systematic parameters as proven later.

In following subsections, we present our theoretical evidence, which was contributed in the publication [57], to explain the speedup of any SDEB implementation in comparison to its counterpart PUND implementation. Under some weak conditions, the results show that the bounds of the speedup

[12]The maximum number of steps that the algorithm uses on any input of the length specified. For the purpose of simplicity, we sometimes call them the conquering-stage runtime function or conquering-stage runtime function instead.

depend on the number of subproblems and on the complexity of the conquering-stage algorithm. The next section presents the lower and upper bounds of the speedup for a class of functions of the conceptual runtime of the conquering-stage algorithm on inputs of a given length. It is next shown that these results of the speedup are still valid for a class of functions extended to almost all practical conquering-stage algorithms in asymptotical case when input size approaches to infinity. Finally, the last section gives conclusions and directions for future works.

## 4.2.2 Speedup of the SDEB approach

### 4.2.2.1 Runtime efficiency of SDEB vs its PUND implementations

Firstly, we consider SDEB implementations whose conquering-stage algorithms conceptually empirical runtime functions belong to the class of functions $\Omega$ described as below:

$$\Omega = \{F : \mathbb{R}^+ \to \mathbb{R}^+ : \ F(x) = \sum_{i=1}^{m} \alpha_i x^{\beta_i}, \ m \in \mathbb{N}, \beta_m > \ldots > \beta_1 > 1, \ \alpha_m \neq 0, \alpha_i \geq 0; \forall i\}. \tag{4.2.1}$$

We then analyze the speedup of a SDEB implementation versus its PUND such that both of them satisfy ( 4.2.1 ).

Secondly, we pay attention to the speedup when the conceptual runtime functions belong to a class of functions $\widetilde{\Omega}$ formed by relaxing the condition on the coefficients $\alpha_i$ in $\Omega$. These coefficients in functions belonging to $\widetilde{\Omega}$ can receive either positive or negative values as long as the highest degree

coefficient $\alpha_m$ is still positive [13].

$$\widetilde{\Omega} = \{F : \mathbb{R}^+ \to \mathbb{R}^+ : \ F(x) = \sum_{i=1}^{m} \alpha_i x^{\beta_i}, \ m \in \mathbb{N}, \beta_m > \ldots > \beta_1 > 1\}.$$

It is obvious that $\widetilde{\Omega}$ is a broader class of functions than $\Omega$. As explained later that approximated runtime functions of practical conquering-stage algorithms are closer to the functions in $\widetilde{\Omega}$ than to those in $\Omega$.

For a given input instance, the following two assumptions are assumed to hold in all Lemmas and Corollaries presented in this section unless otherwise specified.

**Assumptions 1.**

a. *The runtime function of conquering-stage algorithm belongs to* $\Omega$.

b. *The total runtime for the execution of both decomposing and combining stage is much less significant compared to the total runtime for getting all subsolutions (equal to the runtime for the execution of the conquering stage).*

The assumption b. is actually saying that total runtime of a SDEB implementation is most vastly contributed by the conquering-stage algorithm.

Given an SDEB implementation, let $A$ be the runtime of its conquering stage's execution, and $B$ be that of its PUND counterpart. Then the speedup under the above conditions is defined as $\frac{B}{A}$. For the sake of convenience in proofs, we will find the bounds of the fraction $\frac{A}{B}$. Our main result is the following:

[13] We want to stress the point that $\alpha_m$ is always positive since the runtime function $F(x)$ must approaches to $\infty$ when $x$ approaches to $\infty$.

**Theorem 4.2.1.** *The speedup of a SDEB implementation relative to to its corresponding PUND has*

a. *An upper bound of $k^{\beta_m - 1}$, where $k$ is the number of subproblems, $\beta_m$ is the degree of the runtime function $F$ of the conquering-stage algorithm*[14].

b. *A lower bound of* 1*; but a larger lower bound can be obtained by changing some systematic parameters of the decomposing algorithm.*

**Proposition 4.2.2.** *Given $k \in \mathbb{N}, k > 1; d_j > 0, \forall j = \overline{1..k}$ such that $\sum_{j=1}^{k} d_j = 1$ then inequalities:*

$$1 > \sum_{j=1}^{k} d_j^{\beta} \geq \frac{1}{k^{\beta-1}} \tag{4.2.2}$$

*hold $\forall \beta > 1$. The equality takes place if $d_j = \frac{1}{k}\ \forall j$.*

*Proof.* It is clear that $0 < d_j^{\beta} < d_j, \forall j = \overline{1,k}$, sum up these $k$ distinct inequalities side by side then the left hand side of the inequality (4.2.2) is followed. Let $f(x) = x^{\beta}, \beta > 1$, then the $2^{nd}$ derivative of $f$ is $f''(x) = \beta(\beta-1)x^{\beta-2} > 0,\ \forall x > 0$. Thus $f(x)$ is a convex function in $(0, \infty)$. According to Jensen inequality [15], we produce:

$$f\left(\frac{\sum_{j=1}^{k} d_j}{k}\right) \leq \frac{\sum_{j=1}^{k} f(d_j)}{k} \rightarrow \frac{1}{k^{\beta}}\left(\sum_{j=1}^{k} d_j\right)^{\beta} \leq \frac{1}{k}\sum_{j=1}^{k} d_j^{\beta} \rightarrow \frac{1}{k^{\beta-1}} \leq \sum_{j=1}^{k} d_j^{\beta}. \tag{4.2.3}$$

The equality takes place clearly if and only if $d_j = \frac{1}{k}\ \forall j = \overline{1,k}$. □

**Proposition 4.2.3.** *With any numbers $a,\ b,\ c,\ d\ > 0$, we have*

$$\max\left(\frac{a}{b}, \frac{c}{d}\right) \geq \frac{a+c}{b+d} \geq \min\left(\frac{a}{b}, \frac{c}{d}\right). \tag{4.2.4}$$

[14] By degree, we mean the highest power of components in $F$.

[15] `http://mathworld.wolfram.com/JensensInequality.html`.

*Proof.* Because of equivalent role between $\frac{a}{b}$ and $\frac{c}{d}$, do not loose the generality, assume that $\frac{a}{b} \leq \frac{c}{d}$. Then, the inequality (4.2.4) is equivalent to

$$\frac{c}{d} \geq \frac{a+c}{b+d} \geq \frac{a}{b} \leftrightarrow \begin{cases} bc + cd \geq ad + cd, \\ ab + bc \geq ab + ad \end{cases} \leftrightarrow bc \geq ad.$$

□

Both equalities in the inequality (4.2.4) happen if and only if $\frac{a}{b} = \frac{c}{d}$.

**Corollary 4.2.4.** *With any* $a_i,\ b_i > 0\ \forall i = \overline{1,m}$, *then*

$$\max_{i=1,\dots,m}\left\{\frac{a_i}{b_i}\right\} \geq \frac{\sum\limits_{i=1}^{m} a_i}{\sum\limits_{i=1}^{m} b_i} \geq \min_{i=1,\dots,m}\left\{\frac{a_i}{b_i}\right\}.$$

From Proposition (4.2.3) and by induction, it is not difficult to obtain this corollary; these both equalities take place if $\frac{a_i}{b_i} = \frac{a_j}{b_j},\ \forall i \neq j$.

**Lemma 4.2.5.** *If* $m, k \in \mathbb{N}, m \geq 1, k > 1; \beta_m > \ldots > \beta_1 > 1; b_i > 0\ \forall i = \overline{1,m}$ *then*

$$\frac{\sum\limits_{i=1}^{m} \frac{1}{k^{\beta_i - 1}} b_i}{\sum\limits_{i=1}^{m} b_i} \geq \frac{1}{k^{\beta_m - 1}}.$$

*The equality takes place if* $m = 1$.

*Proof.* Deriving from Corollary (4.2.4), we have:

$$\frac{\sum\limits_{i=1}^{m} \frac{1}{k^{\beta_i - 1}} b_i}{\sum\limits_{i=1}^{m} b_i} \geq \min_{i=1,\dots,m}\left\{\frac{\frac{1}{k^{\beta_i - 1}} b_i}{b_i}\right\} = \min_{i=1,\dots,m}\left\{\frac{1}{k^{\beta_i - 1}}\right\} = \frac{1}{k^{\beta_m - 1}}. \tag{4.2.5}$$

Except for $m = 1$, the equality does not take place since the sequence $\left\{\frac{1}{k^{\beta_i - 1}}\right\}_{i=1}^{m}$ is strictly monotonic, so this lemma is proved. □

**The proof of Theorem (4.2.1):**

*Proof.* Let $n$ be the size of the original instance, $c_j$ be the size of the $j^{th}$ subproblem (thus $\sum_{j=1}^{k} c_j = n$), $k$ be the number of subproblems and $F(\cdot) \in \Omega$ be the conquering-stage runtime function, we have the runtime of the execution of the conquering stage:

$$A = \sum_{j=1}^{k} F(c_j) = \sum_{j=1}^{k} \sum_{i=1}^{m} \alpha_i c_j^{\beta_i} = \sum_{i=1}^{m} \left( \sum_{j=1}^{k} \alpha_i c_j^{\beta_i} \right) = \sum_{i=1}^{m} a_i,$$

where $a_i = \sum_{j=1}^{k} \alpha_i c_j^{\beta_i}, \forall i = \overline{1,m}$. Similarly:

$$B = F(n) = \sum_{i=1}^{m} \alpha_i n^{\beta_i} = \sum_{i=1}^{m} b_i,$$

where $b_i = \alpha_i n^{\beta_i}, \forall i = \overline{1,m}$.

Let $d_j = \frac{c_j}{n}, \forall j = \overline{1,k}$, with the note of $n = \sum_{j=1}^{k} c_j$, hence $\sum_{j=1}^{k} d_j = 1$, then according to Proposition (4.2.2), we deduce:

$$1 > \sum_{j=1}^{k} d_j^{\beta_i} = \frac{\sum_{j=1}^{k} c_j^{\beta_i}}{(\sum_{j=1}^{k} c_j)^{\beta_i}} \geq \frac{1}{k^{\beta_i - 1}}, \rightarrow 1 > \frac{\sum_{j=1}^{k} \alpha_i c_j^{\beta_i}}{\alpha_i (\sum_{j=1}^{k} c_j)^{\beta_i}} = \frac{a_i}{b_i} \geq \frac{1}{k^{\beta_i - 1}},$$

for all $i \geq 1$.

Combining with Corollary (4.2.4) and Lemma (4.2.5), we have

$$1 > \max_{i=1,\ldots,m} \left\{ \frac{a_i}{b_i} \right\} \geq \frac{\sum_{i=1}^{m} a_i}{\sum_{i=1}^{m} b_i} = \frac{A}{B} \geq \frac{\sum_{i=1}^{m} \frac{1}{k^{\beta_i - 1}} b_i}{\sum_{i=1}^{m} b_i} \geq \frac{1}{k^{\beta_m - 1}}. \tag{4.2.6}$$

The right-hand-side equality happens when $m = 1$ and $\frac{a_i}{b_i} = \frac{1}{k^{i-1}} \leftrightarrow c_1 = c_2 = .. = c_k = \frac{n}{k}$. □

The above proof concludes the validity of Theorem (4.2.1). The below corollary shows that the lower bound can reach to a higher value than 1 and that higher value depends on the fraction of the maximum size of subproblems over the size of the original instance.

**Corollary 4.2.6.** *Given any $K_{max} \in \mathbb{N}, K_{max} > 1$, if the size of every subproblem is bounded in the interval $[1, K_{max}]$ then*

$$\varepsilon^{1-\beta_1} \leq speedup,$$

*where $0 < \varepsilon = \frac{K_{max}}{n} \leq 1$. The equality happens when $m = 1$ and $k \cdot K_{max} = n$.*

*Proof.* Let

$$d_i = \frac{c_i}{n} \to 1 = \sum_{i=1}^{k} d_i \text{ and } 0 < d_i \leq \varepsilon \leq 1.$$

We have

$$\frac{a_i}{b_i} = \sum_{j=1}^{k} d_j^{\beta_i} \leq \sum_{j=1}^{k} d_j \varepsilon^{\beta_i - 1} = \varepsilon^{\beta_i - 1} \sum_{j=1}^{k} d_j = \varepsilon^{\beta_i - 1}, \tag{4.2.7}$$

for all $i = \overline{1, m}$. The equality takes place if $d_j = \varepsilon, \ \forall j$, hence $k \cdot K_{max} = n$. Replace the inequality (4.2.7) into the left hand side of inequality (4.2.6), the following inequality holds:

$$1 > \varepsilon^{\beta_1 - 1} = \max_{i=1,\dots,m} \left\{ \varepsilon^{\beta_i - 1} \right\} \geq \max_{i=1,\dots,m} \left\{ \frac{a_i}{b_i} \right\} \geq \frac{A}{B}.$$

Since the sequence $\left\{ \varepsilon^{\beta_i - 1} \right\}_{i=1}^{m}$ is strictly monotonic, hence the left-hand-side equality does not take place with except for $m = 1$, or

$$\varepsilon^{\beta_1 - 1} \geq \frac{A}{B} \longleftrightarrow \varepsilon^{1-\beta_1} \leq \frac{B}{A} = \text{speedup}. \tag{4.2.8}$$

□

**Remark 4.2.7.** *Under the conditions stated, the result of the part **a** in theorem (4.2.1) suggests that a SDEB implementation is faster than its PUND counterpart, but cannot be faster than $k^{\beta_m - 1}$ times. This is the ultimate upper limit to the speedup of a SDEB implementation relative to its PUND counterpart. The lower this upper limit, the less runtime efficiency that SDEB implementation gains and vice versa.*

*The result of part* ***b*** *presents a very simple way which allows the speedup to receive an adjustable minimum value by setting a maximum size* $K_{max}$ *for all subproblems. By doing so, one almost guarantees that the speedup cannot be smaller than* $\varepsilon^{1-\beta_1}$ *which is known in advance. In other words, this result also shows an advantage of the SDEB approach in efficiency and effectiveness. For example, we can use this result to predict the speedup before the second stage is performed so that we can reserve ahead a certain amount of time for the combining algorithm, used in the combining stage, to improve the quality of the final solution while still run faster than the PUND counterpart.*

*However, the first condition imposed on the conquering-stage runtime functions is quite restrictive; thus the range of application of this theorem becomes narrower. As we know, conceptual runtime functions of practical conquering-stage algorithms can be represented directly or approximately (by Taylor's expansion) as the function form in* $\widetilde{\Omega}$.

Below is a modified result of theorem (4.2.1) for the broader class of runtime functions $\widetilde{\Omega}$ which seems very close to runtime functions of practical algorithms.

**Lemma 4.2.8.** *Assume* $v = (v_1, \ldots, v_k) \in \mathbb{R}^k_+$, *and* $0 < \alpha < \beta$, *then*

$$\lim_{\|v\| \to \infty} \frac{F_\alpha(v)}{F_\beta(v)} = 0,$$

*where* $F_x$ *belongs to a space on* $\mathbb{R}^k_+, k \geq 1$ *with the maximum norm*[16] *and* $\forall x > 0$, $F_x$ *is determined as*

$$F_x(v) = \sum_{i=1}^{k} v_i^x.$$

[16]Defined as $\|v\| = \max_{i=\overline{1,k}} \{v_i\}$.

*Proof.* We have, with $0 < \alpha < \beta$:

$$0 \leq \lim_{\|v\|\to\infty} \frac{F_\alpha(v)}{F_\beta(v)} = \lim_{\|v\|\to\infty} \left\{\frac{\|v\|^\alpha}{\|v\|^\beta}\right\} \left\{\frac{c + \sum\limits_{v_i<\|v\|} \frac{v_i^\alpha}{\|v\|^\alpha}}{c + \sum\limits_{v_i<\|v\|} \frac{v_i^\beta}{\|v\|^\beta}}\right\} = 0 \rightarrow \lim_{\|v\|\to\infty} \frac{F_\alpha(v)}{F_\beta(v)} = 0. \quad (4.2.9)$$

Due to $\sum\limits_{i=1}^{k} v_i$ is the Manhattan norm on that space, so ( 4.2.9 ) still holds if we replace the maximum by the Manhattan norm. In other words:

$$0 = \lim_{\left(\sum\limits_{i=1}^{k} v_i\right)\to\infty} \frac{F_\alpha(v)}{F_\beta(v)} = \lim_{\left(\sum\limits_{i=1}^{k} v_i\right)\to\infty} \frac{\sum\limits_{i=1}^{k} v_i^\alpha}{\sum\limits_{i=1}^{k} v_i^\beta}. \quad (4.2.10)$$

□

Let us explain the meaning of ( 4.2.10 ). If we consider $v_i$ as the size of $i^{th}$ subproblem, $\sum\limits_{i=1}^{k} v_i$ as the size of the original instance, then the computational time associated with the $i^{th}$ coefficients $\alpha_i$ in a function of $\Omega$ is $a_i = \alpha_i \cdot \sum\limits_{i=1}^{k} v_i^{\beta_i}$. Lemma (4.2.8) determines that if the size of original instances is large enough, then the computational time associated with $\alpha_i \ \forall i < m$ can be ignored. In other words, if the condition on the "positive" sign of those lower degree coefficients in the definition of $\Omega$ is relaxed, then when the size of original instances is very enough, we can remove those low-degree components out of the conceptual runtime functions without impacting much on the value of functions; or mathematically, the form of the conceptual runtime functions can be reduced to $F(x) = \alpha_m \cdot x^{\beta_m},\ \alpha_m > 0, \beta_m > 1$. In consequence, the lower bound of the speedup in this *asymptotical case* [17] will be $\frac{1}{c^{\beta_m - 1}}$ which is greater than that shown in the left-hand-side of the inequality (4.2.8) since $1 \leq \beta_1 < \beta_m$. Hence, theorem (4.2.1) in the asymptotical case is briefly pre-

[17] The case when the input size approaches to infinity.

sented as:

$$\frac{1}{\varepsilon^{\beta_m - 1}} \leq speedup \leq k^{\beta_m - 1}, \tag{4.2.11}$$

where $\varepsilon$, $k$ are defined in theorem (4.2.1) and corollary (4.2.6).

#### 4.2.2.2 Difference between our findings and Amdahl's law

Our findings should not be confused with Amdahl's law [3] which is fundamentally different from ours though both of them deal with things related to speedup. Indeed, Amdahl's law governs the speedup in solving a problem using *parallel processors* in comparison to using only one serial processor to solve the same problem. According to this law, it is possible to derive an upper bound of the speedup when the number of processors approaches to infinity. In contrast to Amdahl's law, our results deals with the speedup of a SDEB implementation compared to its PUND version. Moreover, the two implementations in our case are executed on the same serial processor. As shown in the previous subsection that the speedup is bounded for given inputs of given size.

#### 4.2.2.3 Discussion

This analytical study investigates the bounds on the runtime speedup of SDEB implementation with reference to its PUND counterpart. It has been proven mathematically that the speedup is bounded. Given an input instance, the upper bound of the speedup can be determined by the degree of the conceptual runtime function of the conquering-stage algorithm and the number of subproblems. However, it is not proven here that the bound is a *strict upper bound* in a general case except in the asymptotical case. In

asymptotical case, that upper bound is actually the strict bound and in order to reach a speedup as high as possible, all subproblems must have same size[18].

Under the assumption **1.b** of runtime of the dividing and combining stages, the lower bound is always greater than 1 implying that a SDEB implementation runs faster than its PUND counterpart. This lower bound can be increased by setting a maximum number of *elements* [19] for subproblems as proven in Corollary 4.2.6. For the asymptotical case that when size of the original instance is very huge, according to the inequality (4.2.11) in Lemma (4.2.8), the lower bound increases even without changing that maximum number of elements.

Our results are different from Amdahl's law not only because of serial versus parallel realization but also for different bound values. The bounds in our investigation depend on input size, parameters settings, and complexity of the conquering-stage algorithm, whereas in Amdahl's law the bound values do not depend on any of these factors. Moreover, Amdahl's law shows the finite upper bound of speedup of any program run on parallel machines no matter how many processors are used; in contrast, we showed that the speedup of SDEB implementation is bounded but its upper bound $k^{\beta_m - 1}$ increases if the number of subproblems increase, and this bound is surely finite only if the number of subproblems $k$ is finitely bounded. It should be cautious

[18]Since the reduced form of conceptual runtime function in $\widetilde{\Omega}$ indicates $m = 1$, the equal-in-size condition for all subproblems makes all equalities take place in Propositions, Lemmas, and Corollaries (in previous subsection).

[19]Which means *cities or nodes* for TSP and/or VRP.

when applying the bounds if the assumption of running time of conquering stage to be greater than other stages is violated. From current analytical results, the running time of a SDEB implementation becomes shortest when all sub-problems are at equal size. As decomposing and combining stage can be independent of conquering stage which lead to their running time might be independent of sub-problem size, the assumption will be hence violated if the decomposing and combing stage produce longer running time than conquering stage when all sub-problems are at equal size. Finally, by assigning an available processor to the task of solving a subproblem, we can easily build up an efficient and effective parallel version of a serial SDEB implementation. We would like to point it out here that the result presented here should be interpreted cautiously as the upper bound has not been proved to be the strict upper bound of the speedup. We expect to address that issue in future.

Decomposing algorithms are usually defined based on the so-called "distance function" which measures the distance between two points in search space. For example, in Euclidean Traveling Salesman Problem (ETSP), this function is the Euclidean distance function. But the definition of a suitable distance function is problem-dependent and is aimed at describing as close as possible the impact (or role) of one or more components of a solution[20] upon the complete solution of the problem. On the other hand, not less important than building up a suitable distance function, decomposing algorithms should be designed such that 1) it is convenient to re-construct a high quality solution to the original instance from the subsolutions and 2) it results in

[20]Components are, for example, two joining cities in a closed tour of TSP.

subproblems of approximately equal size to gain better speedup.

These findings have shown the runtime efficiency of the SDEB approach in comparison to its PUND approach. One may question about the quality of solution of the SDEB approach compared to that of the PUND. In fact, the solution quality of the SDEB approach depends not only on the *strength* of the conquering-stage algorithm but also on how well decomposing and combining algorithms are designed and how skillful the combination between them is. Consequently, structure of problem will affect on the way of combining subsolutions of and designing those algorithms. If the structure of problem can be exploited to ease the problem decomposition into subproblems such that structure of these subproblems are highly independent of each other, then an SDEB approach which makes full use the problem structure is a promising candidate to solve the problem effectively.

Comparing the solution quality between an SDEB and PUND approaches in general is not possible since the design of decomposing and combining algorithms is problem-dependent i.e. there is no general paradigm for creating those algorithms. But, given a particular problem, this comparison can be done in some ways. One possible way is to determine deterministically/probabilistically whether or not at least one optimal solution exists in the search space of the SDEB implementation. Another way is to prove on the convergence to optimality of the SDEB implementation. Following the former way, we obtained a sufficient condition for instances of Euclidean TSP that ensure that all optimal solutions are in the search space by a simple decomposing and combining algorithms. This finding will be presented in subsection 4.3.

#### 4.2.2.4 SDEB in relationship with POPMUSIC and Dantzig-Wolf Principe - How many subproblems need to be solved?

There are some similar characteristics shared between our SDEB approach and POPMUSIC. Firstly, in POPMUSIC a search procedure is used to optimize the sub-problems formed by parts resulted from a clustering algorithm rather than optimize the original problem. This procedure can be viewed as playing the same role as the conquering-stage algorithm in SDEB approach. Secondly, if $r = p$ in POPMUSIC the implementation is equivalent to the PUND employing that search procedure. Thirdly, after a sub-problem is locally optimized, there is a certain procedure used to recover the solution of the original problem from that of the sub-problem. In SDEB methods, such a procedure appears in the combining stage.

But there are some dissimilarities between them. Indeed, the SDEB approach may require the sub-problems right after clustering stage and before the execution of the conquering stage to be known, whereas POPMUSIC builds such a sub-problem from the parts. In addition, sub-problems in SDEB are distinct from each other, but in POPMUSIC they can interfere mutually. In SDEB, the number of sub-problems are fixed, whereas this quantity in POPMUSIC cannot be known until the end of the execution.

From the runtime efficiency point of view, the results of the speedup bounds presented in section 4.2 can be applied to analyze runtime performance of POPMUSIC. More specifically , if the sum of the sizes of all sub-problems, which would be optimized in iterations of POPMUSIC's implementation, is close to the size of the original instance, then our results of speedup bounds can be used to approximate the speedup of the POPMUSIC

and its PUND version. A similar reasoning is also applicable to Dantzig-Wolf Principle.

Now, assume there is an implementation of these methods to solve a certain COP and this implementation will eventually end its solving process after solving $K$ subproblems with corresponding size $N_i, \forall i = \overline{1, K}$ for an optimization problem instance with size $N$. Set

$$M = \sum_{i=1}^{K} N_i.$$

Devised by runtime analysis in the previous section, we will try to answer how many subproblems this implementation should run to maintain a better performance in terms of runtime in comparison to its PUND version. Assume that the empirical runtime function of the algorithm used to solve subproblem belongs to $\Omega$ (see Equation (4.2.1)) we have:

$$\frac{1}{\epsilon^{\beta_1 - 1}} \geq \frac{\sum\limits_{i=1}^{K} F(N_i)}{F(\sum\limits_{i=1}^{K} N_i)} = \frac{\sum\limits_{i=1}^{K} F(N_i)}{F(M)} \geq \frac{1}{k^{\beta_m - 1}} \tag{4.2.12}$$

where

$$\epsilon = \frac{\max\limits_{i=\overline{1,K}} N_i}{M},$$

and empirical runtime of that implementation is

$$A = \sum_{i=1}^{K} F(N_i)$$

Let $B = F(N)$ be the empirical runtime of the PUND version of that implementation on the original instance, inequalities (4.2.12) are equivalent to

$$\frac{1}{\epsilon^{\beta_1 - 1}} \geq \frac{A}{F(M)} = \frac{A}{B}\frac{F(N)}{F(M)} \geq \frac{1}{k^{\beta_m - 1}} \Rightarrow \frac{1}{\epsilon^{\beta_1 - 1}}\frac{F(M)}{F(N)} \geq \frac{A}{B} \geq \frac{1}{k^{\beta_m - 1}}\frac{F(M)}{F(N)} \tag{4.2.13}$$

From Corollary 4.2.4, we deduce the following claim:

$$\min_{j=\overline{1,m}}\left(\frac{M}{N}\right)^{\beta_i} \leq \frac{F(M)}{F(N)} = \frac{\sum\limits_{j=1}^{m}\alpha_j M_j^{\beta}}{\sum\limits_{j=1}^{m}\alpha_j N_j^{\beta}} \leq \max_{j=\overline{1,m}}\left(\frac{M}{N}\right)^{\beta_i}. \tag{4.2.14}$$

**Remark 4.2.9.** *In the event that total size of solved subproblems is smaller or equal to the size of the original instance, i.e.* $M \leq N$*, then that decomposition-based implementation still runs faster than its PUND version at least* $\epsilon^{\beta_1-1} * \left(\frac{N}{M}\right)^{\beta_1}$ *times due to*

$$\max_{j=\overline{1,m}}\left(\frac{M}{N}\right)^{\beta_i} = \left(\frac{M}{N}\right)^{\beta_1} > \left(\frac{M}{N}\right)^{\beta_2} > \ldots \left(\frac{M}{N}\right)^{\beta_m} = \min_{j=\overline{1,m}}\left(\frac{M}{N}\right)^{\beta_i}.$$

*Moreover, an upper bound of the speedup between runtime of that decomposition-based implementation to runtime of its PUND version is derived as*

$$speedup = \frac{B}{A} \leq k^{\beta_m-1}\left(\frac{N}{M}\right)^{\beta_m}$$

*where the equality takes place if and only if* $m = 1$*. When* $m > 1$*, this upper bound is no longer the tightest bound of the speedup. That bound value becomes the tightest bound only when* $M = N$*. This implicates that, we can regard the above bound value as a good approximate upper bound when* $M$ *is close to* $N$*. When the total size of subproblems is much far smaller than the size of original instance, the approximation for the upper bound here will become less accurate than when the total size is very close to the original instance's size.*

**Remark 4.2.10.** *When total size of all solved subproblems is greater than the size of the original instance, i.e.* $M > N$*, similarly we have:*

$$\min_{j=\overline{1,m}}\left(\frac{M}{N}\right)^{\beta_i} = \left(\frac{M}{N}\right)^{\beta_1} < \left(\frac{M}{N}\right)^{\beta_2} < \ldots \left(\frac{M}{N}\right)^{\beta_m} = \max_{j=\overline{1,m}}\left(\frac{M}{N}\right)^{\beta_i}.$$

*Hence,*

$$\epsilon^{\beta_1-1} * \left(\frac{N}{M}\right)^{\beta_m} \leq speedup = \frac{B}{A} \leq k^{\beta_m-1}\left(\frac{N}{M}\right)^{\beta_1}$$

*If $M$ is large enough such that the right-most-hand-side term is less than* 1, *i.e. the decomposition-based implementation needs to solve too many subproblems before reaching a good solution (to the original instance), then the decomposition-based implementation is no longer more efficient in term of runtime perspective than the non-decomposition-based version. More precisely, if $M > M_0$ where*

$$M_0 = N * \sqrt[\beta_1]{k^{\beta_m - 1}}.$$

*then the decomposition-based implementation becomes less efficient than its non-decomposition-based version.*

## 4.3 On the Optimality of Solutions to Euclidean TSP using a simple SDEB Method

As discussed in subsection 4.2.2.3 on pros and cons of SDEB methods, besides efficiency in runtime there is an obstacle for SDEB methods in achieving high quality solutions that lies at the decomposing and combining stages. To ensure that achievement for a certain SDEB implementation over its PUND implementation, we need to guarantee that there is always a "positive chance" of finding optimal solutions for the SDEB. Solution quality of that SDEB implementation will depend on how strong the guarantee is. The most simple guarantee is to ensure that the search space of that SDEB implementation will contain at least one optimal solution[21]. Another guarantee that is harder to obtain is to ensure the convergence to optimal solutions for the SDEB implementation. In this subsection, we will conduct a case study on ETSP using a

[21]There is possibility that optimal solutions are filtered out of the search space due to decomposing and combining procedures.

simple SDEB method. The aim of the study is to obtain a sufficient condition on structure of ETSP to guarantee that the SDEB method will have optimal solutions in its search space.

The work in this subsection contributed a part in the publication [56]

### 4.3.1 The Optimal Solutions in Solution Space - a Sufficient Condition on Structure of ETSP

In this paragraph, we consider the case where a SDEB method is applicable to ETSP. In accordance to SDEB approaches' scheme in section 4.2, after obtaining all solutions to subproblems, one needs to assemble them to form a feasible solution, i.e. a *closed tour* in the context of solving TSP, to the original instance. The most intuitive approach is to choose some edges on tours of subproblems (we will use the term "sub-tour" onwards instead) then create *bridges* linking sub-tours mutually (these bridges will have vertices from chosen edges) in such a way that after those chosen edges are removed from sub-tours, bridges and the rest edges of sub-tours will form a closed tour - a feasible solution to the original instance. If we search for such a feasible solution by increasing the number of chosen edges of all sub-tours to the number of edges of corresponding sub-tours then this procedure will be intractable as the ETSP is a NP-hard problem [5]. So, it is the best choice if we can confine the number of chosen edges (for each sub-tour) to one edge. However, one of possible concerns such as whether or not optimal solutions are in the search space with such a limitation, may be raised.

This concern will be resolved by a sufficient condition on ETSP's structure in this subsection. If any ETSP instance satisfies this condition, then all opti-

mal solutions of this instance must be in the search space of a certain SDEB approach.

**Theorem 4.3.1.** *Assume the number of subproblems equals two, and let A and B denote these two set of points*[22]*, $d_A$ and $d_B$ be diameter*[23] *of A and B, respectively, and $d_{AB}$ be distance*[24] *between A and B. If there exist $n \in N, 2 \leq n \leq \min\{|A|, |B|\}$ such that*

$$d_{AB} \geq d_n = \frac{\min\{d_A, d_B\}}{2(n-1)} + d_A + d_B, \tag{4.3.1}$$

*then there exists an SDEBimplementation whose search space contains all optimal solutions to the original instance and that search space is formed with a total number of chosen edges of two sub-tours less than 2n.*

*Proof.* Let $T_n$ be the set of all feasible tours to the original instance obtained by the combining procedure from two tuples. Where each tuple has $n$ edges chosen from a part (the number of edges of a tuple is equal to the other's to make a feasible tour to the original instance). Obviously we have $1 \leq n \leq \frac{\min\{|A|,|B|\}}{2}$. Set $S_n = \min\{\text{cost of a tour t} = |t| : t \in T_n\}$, we will prove that if ( 4.3.1 ) holds then

$$S_n > S_1 \ \forall n > 1. \tag{4.3.2}$$

We choose any $n$ edges for each sub-tour whose lengths are $a_i, b_i, i = \overline{1..n}$, where $a_i \in A, b_i \in B \ \forall i$. Let $x_i, i = \overline{1..2n}$ be the length of the $i^{th}$ bridge, $\xi$ be the

[22]Each set contains all points of one subproblem

[23]By diameter of a set, we mean the longest distance between any two points of the set.

[24]By distance between two sets, it is defined as the shortest distance between any pair of points, one from a set, another from the other one.

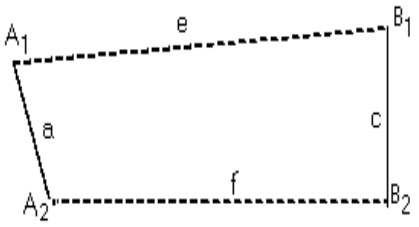


Figure 4.2: There are at least two chosen edges, one from a sub-tour, the other from the other one; such that the vertices of two bridges $A_1B_1$ and $A_2B_2$ are from their four vertices $A_1, A_2, B_1, B_2$ only. Here $a, c, f, e$ are lengths of edges accordingly.

sum of length of the rest edges of the two sub-tours (not chosen edges) . We need to consider following two cases:

**Case 1:** There exists at least one pair of edges such that the two bridges formed from only their four vertices as shown in Fig.4.3.1. So, the length of the final tour is

$$|t_2| = \xi + \sum_{i=1}^{2n-2} x_i - \sum_{i=1}^{n-1} (a_i + b_i) + (e + f - a - c).$$

From ( 4.3.1 ), we have

$$\sum_{i=1}^{2n-2} x_i \geq (2n-2)d_{AB} > 2(n-1)(d_A + d_B) \geq 2\sum_{i=1}^{n-1} (a_i + b_i)$$

$$\rightarrow |t_2| > \xi + \sum_{i=1}^{n-1} (a_i + b_i) + (e + f - a - c) \geq S_1.$$

**Case 2:** If case 1 does not happen, we can assume without loss of generality that $d_A \geq d_B$ and there are two edges of the sub-tour of $A$ and two of $B$ with bridges formed as shown in Fig.4.3. The length of the final tour, in this case, is

$$|t_2| = \xi + \sum_{i=1}^{2n-4} x_i - \sum_{i=1}^{n-2} (a_i + b_i) + (e + g + h + i - a - b - c - d).$$

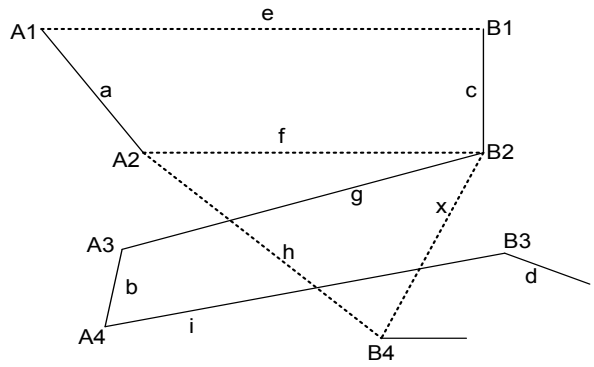


Figure 4.3: There is no existence of any two chosen edges, one from a sub-tour, another from the other one; such that the vertices of two bridges come from their four vertices. Here, $A_1A_2$, $A_3A_4$ are chosen edges of cluster $A$ and $B_1B_2$ is a certain chosen edge of cluster $B$. Lowercase letters stand for the lengths of according edges. Since $A_1$ links to $B_1$, so $B_2$ must links to $A_3$ (cannot be $A_2$ due to the assumption), then $A_4$ definitely links to another node, name $B_3$ and so on ...

Due to $h + x > f$ and from the assumption ( 4.3.1 ), we have

$$\sum_{i=1}^{2n-4} x_i + (g + i) \geq d_B + 2(n-1)(d_A + d_B) \geq 2\{\sum_{i=1}^{n-2} (a_i + b_i) + (b + d)\} + x.$$

$$\rightarrow |t_2| > \xi + \sum_{i=1}^{n-2} (a_i + b_i) + (b + d) + (e + f - a - c) \geq S_1$$

Hence if this case takes place, one always has another way of choosing one edge only from each sub-tour , and this choosing will definitely form a certain tour belonging to $T_1$ such that the length of this new formed tour is absolutely lower than the length of the tour in $T_n$ (in this case this length equals $|t_2|$).

In other words, these both cases have proven successfully the inequality (4.3.2). Since

$$d_n = \frac{\min\{d_A, d_B\}}{2(n-1)} + d_A + d_B$$

is a monotonically decreasing sequence, if ( 4.3.1 ) takes place at $n = n_0$ then

$$d_{AB} \geq d_{n_0} > d_n \ \forall n > n_0 \rightarrow S_n > S_1 \ \forall n > n_0.$$

In other words , the inequality (4.3.2) is held $\forall n \geq n_0$ or all optimal tours of the original instance are in the search space formed with a number of chosen edges from each sub-tour which is less than $n_0$. Theorem (4.3.1) has been proved. □

**Remark 4.3.2.** *A clear corollary resulting from this theorem is that if $n = 2$ and ( 4.3.1 ) still holds then since the total number of chosen edges for these two sub-tours must be even and less than $2n$, the optimal tours of the original instance will be included in the search space formed by using only two bridges (i.e. choose only one edge from each sub-tour to make bridges). In consequence, the complexity of finding the best tour for the original instance given two sub-tours will be $O(m)$ , where m is the size of the original instance.*

## 4.4 A hybrid SDEB method with ACO for large ETSP

The result in the previous subsection may be theoretically interesting but its assumptions are less frequently met in real-world instances, such as, it is hard to satisfy the condition (4.3.1. The following results are for the case when the number of subproblems is higher than two.

Let us state the following assumptions before entering details. If the number of subproblems $k > 2$, and the inter-distances among them (possibly used interchange as "clusters" or "set of points") are much larger than the diameter of any cluster (this could be the case when clusters are very far away from each other); and the distances among any three clusters meet the triangular inequality; then we have a result stated in Theorem (4.4.1). Noticing that each cluster has an even number of bridges in order to guarantee that the final tour (tour to the original instance) is closed, we obtain:

**Theorem 4.4.1.** *In an optimal final tour, there exist no two clusters that are joined together by at least two bridges.*

*Proof.* Assuming there exists a final tour containing two certain clusters $A$ and $B$ which are joined together by at least two bridges as shown in Fig.4.4. In this figure, the two clusters are joined together by two bridges $AaB$ and $AxB$ whose length are $a$ and $x$, respectively[25]. Due to the fact that there are more than 2 clusters, we can assume without loosing generality that cluster $B$ has at least one bridge of length $y$ which links $B$ to a cluster $C$. By the triangular inequality, $x + y > z$, so if we replace bridges $AB = x$ and $BC = y$ of this final tour by a new bridge $AC = z$ then we obtain a new final tour $t$ which is shorter than the previous tour[26]. In other words, the previous tour cannot be an optimal tour , or equivalently, the optimal final tour cannot contain two bridges used to join two certain clusters together or this theorem is proven. □

[25]For simple visualization, a cluster is drawn as a point in the figure.

[26]It is easy to show $t$ is a feasible solution.

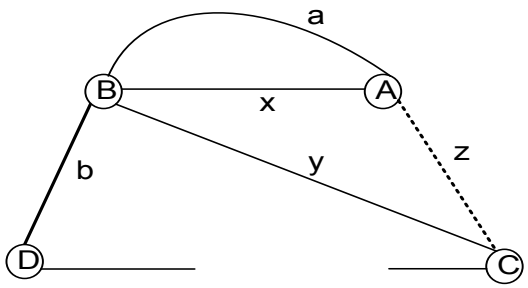


Figure 4.4: Cluster $A$ has two bridges which link $A$ to cluster $B$ with the length of $a$ and $x$, respectively. Here, bridges $AB$ and $BC$ are replaced by $AC$ to get a better solution.

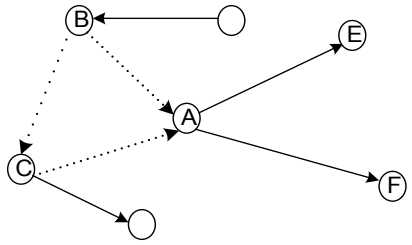


Figure 4.5: Cluster $A$ has at least four bridges linking to other clusters. Bridges $BA$ and $CA$ are replaced by $BC$ to obtain a better solution. The nodes' order of the previous tour can be ..BAF..CAE..B. After being replaced, this order will become ..BC..FAE..B.

**Theorem 4.4.2.** *In an optimal final tour, there exists no cluster that has at least four bridges to link to other four distinct clusters.*

*Proof.* Assuming that the cluster $A$ has at least four bridges, each links to another cluster (according to Theorem (4.4.1)) and the shape of linking is shown in Fig.4.5[27]. Similar to the proof of Theorem (4.4.1), we replace two bridges

[27]The arrows drawn in figure are for the purpose of simpler visualization.

*BA* and *CA* by *BC* and obtain a new final tour which is surely shorter than the old tour. Theorem (4.4.2) is proved. □

Observe that from Theorems (4.4.1, 4.4.2), in order to build an optimal tour from sub-tours, each cluster should have only two bridges which are deployed to link to two different clusters.

To summarize, we have suggested in this subsection a procedure based on Theorems (4.4.1, 4.4.2) to resolve the case that the number of subproblems is more than just two. This case is more frequently met in real-world instances than the case of two clusters as in Theorem (4.3.1). Although this procedure is basically a greedy algorithm and may not result in good solutions for all real-world cases[28], it is possibly effective to (at least) clustered ETSP instances.

## 4.4.1 Experimental Results

### 4.4.1.1 Large scale TSP instances for testing

We test the effectiveness of the proposed SDEB method by applying the algorithm to two different benchmark problems. The first problem is the TSPLIB at http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95. We consider some large Euclidean instances with the number of cities between 650 and 3795. The second benchmark is a group of instances where cities are randomly clustered distributed on the square $[0, 10^6]$. These instances with the number of cities between 1000 and 5000 were generated by the Instance Generator Code of the $8^{th}$ DIMACS Implementation Challenge[29].

[28]Due to No Free Lunch Theorem, this remark is also true for other approaches.

[29]http://www.research.att.com/ dsj/chtsp/download.html

#### 4.4.1.2 Performance comparison between SDEB-ACS and ACS

We compare the proposed SDEB-ACS with ACS. Because the run-time quantity is also used to compare their computational efficiency, both algorithms were run on the same computer (a Dell PC Pentium IV 2.4GHz processor, 256MB of RAM), the same code to implement ACS (except code parts particularly designed for SDEB-ACS including clustering and combining parts), and the same settings for parameters as discussed in [65], $m = 10, \beta = 2, q_0 = 0.98, \alpha = \rho = 0.1$, and $\tau = 1/(n \cdot L_{nn})^{-1}$. In addition, all testing instances are very large, thus for both case, a candidate list is used with the length of $cl = 20$.

#### 4.4.1.3 Memory storage requirement:

For the large-scale TSP instances, the required memory to store the pheromone and cost matrix may contribute most remarkably to the memory requirement of the algorithm. With the input size of N cities, the amount of memory to store these two matrices should be $CN(N-1)$ bytes, where $C$ is a system-dependent parameter to store a real-type number and these two matrices store the upper triangle of matrices only. However, the proposed SDEB method takes, during the execution of ACS, approximately $Cn(n-1)$ bytes where $n$ is the size of largest cluster. It can be seen that, from the point of view of memory requirement, the SDEB-ACS is more efficient than ACS.

#### 4.4.1.4 Experimental results:

To compare the run-time of SDEB-ACS with ACS, we take the factor total run-time of ACS of SDEB-ACS. The larger this factor, the faster the SDEB-ACS is. Due to the fact that the optimal solutions of generated clustered Euclidean

TSP instances are unknown, we use a factor *relative performance* to compare ACS's performance with SDEB-ACS on quality of solutions. This factor (relative performance) is computed by taking proportion of the subtraction of the smallest cost found by ACS with SDEB-ACS's over SDEB-ACS's. Each instance was run totally 15 trials, each trial had 5000 iterations, and for clusters whose size is less than 150 we set the parameter $cl = 0$, because such a cluster may not be considered as a large instance.

**4.4.1.4.1 For clustered Euclidean TSPs:** As shown in the table 4.4.1.4.1, SDEB-ACS always produces better solutions than the standard ACS, in much shorter time. For example, it took 16354.6 seconds for solving a 5000-city instance using ACS, but only 1383.31 seconds for the proposed SDEB-ACS algorithm. This is attributed to the fact that SDEB-ACS outperforms ACS in clustered Euclidean TSP. Moreover, notice that the order of runtime function of ACS $\beta_m = 3$, and according to theorem ( 4.2.1 ) the speedup gained by using ACS as conquer algorithm would not be greater than $k^{\beta_m - 1} = k^2$ where $k$ is number of subproblems. The experimental result in table 4.4.1.4.1 therefore conforms to theoretically estimated bounds.

**4.4.1.4.2 For benchmark Euclidean TSPs:** Because almost large benchmark instances do not follow the sufficient condition mentioned in Theorems (4.4.1, 4.4.2), thus the combining algorithm used for such an instance has a little but important change which takes a part in improving the quality of final solution. After doing the same thing as done for the above type of instance, the "representative tour" (found at step 2 in the above remark) is replaced by a *closed tour* which characterizes like a TSP tour -starts from a city, visit all

Table 4.1: A comparison of SDEB-ACS and ACS is based on clustered instances of 1000-5000 cities randomly generated. Each trial was stopped after 5000 iterations. Averages are over 15 trials. Results in bold are the best in the table. (*) is the proportion of run-time of ACS to SDEB-ACS's; $k$ is the number of clusters. Entries in the results are in Euclidean distance.

| No. of | SDEB | | | | ACS | | | (*)ACS/ | [(2)-(1)] |
|---|---|---|---|---|---|---|---|---|---|
| cities | average | std dev | best$^{(1)}$ | k | average | std dev | best$^{(2)}$ | SDEB | /(1) |
| 1000 | **11870553.80** | 74110.93 | **11747792** | 6 | 12302227.67 | 88129.51 | 12168397 | 3.68 | 3.58% |
| 1500 | **13840435.67** | 70049.18 | **13735671** | 6 | 14143592.20 | 104250.71 | 13971107 | 5.63 | 1.71% |
| 2000 | **16435565.40** | 82162.62 | **16307804** | 8 | 17124049.13 | 187148.32 | 16908891 | 5.08 | 3.69% |
| 2500 | **17841082.93** | 68272.57 | **17740618** | 13 | 18592455.20 | 185965.00 | 18324584 | 6.35 | 3.29% |
| 5000 | **25300826.40** | 82162.62 | **25147562** | 26 | 26515075.20 | 239561.92 | 26202524 | 11.82 | 4.20% |

Table 4.2: A comparison of SDEB-ACS and ACS is based on large benchmark instances. Averages are over 15 trials. Each trial were stopped after 5000 iterations. Results in bold are the best in the table. (*) is the proportion of run-time of ACS to SDEB-ACS's; $k$ is the number of clusters.

| prob. | SDEB | | | | ACS | | | Optimum | (*)ACS/ | [(2)-(1)] |
|---|---|---|---|---|---|---|---|---|---|---|
| name | average | std dev | best$^{(1)}$ | k | average | std dev | best$^{(2)}$ | known | SDEB | /(1) |
| p654 | **35554.67** | 297.48 | **35113** | 3 | 35860.67 | 438.98 | 35120 | 34643 | 1.96 | 0.02% |
| fl3795 | **30804.33** | 120.27 | **30689** | 4 | 30881.33 | 64.97 | 30842 | 28772 | 2.44 | 0.50% |

other cities and come back the starting city- but a city in the *closed tour* can be visited more than once as long as its total length is less than the replaced one. As shown in Table 4.4.1.4.2, SDEB ACS outperformed ACS for both benchmark instances *p654* and *fl3795* both average and optimal solution, but for *fl3795* the ACS seems more stable than SDEB ACS which can be seen from the value of standard deviation.

#### 4.4.1.5 Discussions

The SDEB ACS proposed in this article shows promising results in increasing efficiency both in run-time and quality of solution for large clustered Euclidean TSP. The proposed method runs faster than the conventional ACS

that does not use clustering. Moreover, the proposed algorithm can be converted to a parallel version with a little changes in its serial version. The results presented in this article underscores the idea that both decreasing runtime and guaranteed quality of solutions is achievable when a decomposition-based method is applied to problems whose solutions can be decomposed into sub-solutions.

## 4.5 A One-Level Partitioning-based Implementation for 2D Protein Folding Problem

A protein's spatial structure determines its biological function which is very important in modern molecular biology. However, it is well-known that problem of Protein Structure Prediction (PSP) is intractable on even simple lattice models [55]. This leads to the situation that metaheuristic optimization methods tend to be the most appropriate algorithmic choice to solve PSP. In this section, we conducted a simple experiment showing how to solve PSP for 2D HP models by a GA-based algorithm that uses the idea of reducing the size of search space from decomposition-based search methodology. A new way of representing conformations on 2D grid which reduces the size of search space at least twice in comparison to the size of search space formed by the conventional way, is suggested and experimented in this section. The reasons that we experiment this problem with GA but not ACO are 1) to understand why GA is outperformed by ACO [180] by focusing on the issue of GA's problem representation; 2) to apply the idea of decomposition-based search into GA.

### 4.5.1 The HP Model

The hydrophobic-hydrophilic model (HP model) is one of the most studied simple protein folding models proposed in [55]. HP model is based on the generally accepted assumption that the hydrophobicity of amino acids is one of central forces involved in driving proteins to adopt a compact globular form. The native structure of most proteins contains a hydrophobic core, i.e., the more hydrophobic (non-polar) amino acids are concentrated in compact cores while hydrophilic (polar) amino acids are located on the surface of the protein. Thus, a protein is modeled as a string over $H, P$, where $H$ represents a hydrophobic amino acid and $P$ represents a hydrophilic amino acid. It is clear that the HP model restricts the space of conformations to self-avoiding paths on the lattice in which vertices are labeled by the amino acid.

Scoring in the HP model is based on topological hydrophobic contacts. To evaluate a particular conformation in the HP model, the number of H-H topological contacts in the lattice is counted. A H-H topological contact is formed by a pair of amino acids which are adjacent on the grid and not consecutive in the sequence, see Fig. (4.6). Such a H-H contact is assumed to provide an energy contribution of $-1$. The optimal conformation is the one with the most H-H contacts. As the length of amino acids sequence increases, exhaustive searching of all possible conformations, even on the two-dimensional lattice, becomes intractable. In this case, optimization (or search) algorithms must be used to find near-optimal conformations. Various advanced computational methods have been employed, including evolutionary computation, simulated annealing, Monte Carlo methods, branch and bound and machine learning approaches.

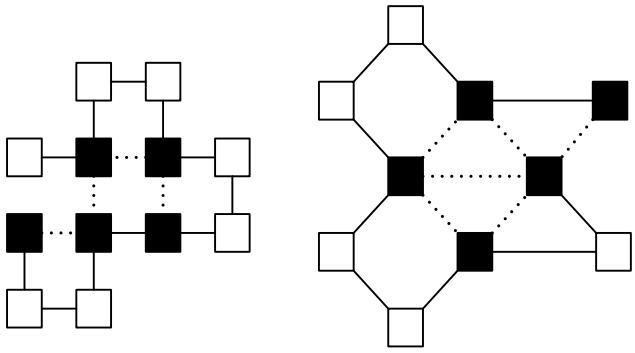

Figure 4.6: HP sequences embedded in the square lattice (the left figure) and the triangular lattice (the right figure). The black squares stand for residues H, while the white for amino acids P. The dot lines show the formed H-H contacts.

Details of setting up a simple GA to solve this problem for a 2D model can be found in the next section. Section 4.5.3.2 will be devoted for experimental results and discussion as well. The last section will give some conclusions about this work.

## 4.5.2 Algorithm Design

When working with lattice models, proteins are often represented using internal coordinates including Relative Encoding and Absolute Encoding. In our algorithm, the relative encoding scheme is used. Furthermore, the HP model under consideration is on the two-dimensional square lattice.

**4.5.2.0.1 Mutation on the Relative Encoding** The effect of one-point mutation on the structure is examined in Fig. (4.7(a)). Its relative encoding is $S_{rel} = FRFRRLFLF$ when viewed from the amino acid marked by 1. A one-

point mutation in the fifth position could produce either of $S^1_{rel} = FRFRFLFLF$ or $S^2_{rel} = FRFRLLFLF$, which are shown in Fig. (4.7(b)) and Fig. (4.7(c)) respectively. From this example, we can see that a one-point mutation in the

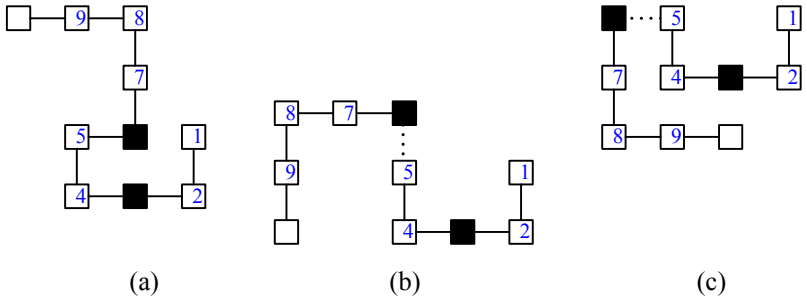


Figure 4.7: In (b) a one-point mutation of the structure in (a) at the fifth gene. An 'R' was mutated to an 'F' producing a lever effect of 90 degrees counterclockwise. In (c) an 'R' was mutated to an 'L' producing a lever effect of 180 degrees counterclockwise. The dot lines in (b) and (c) represent the "mutated" contacts.

relative encoding produces a rotation effect in the structure at the mutated point. To get the same effect in the absolute encoding, one must perform a specific mutation operator so-called *macro-mutation*, that is, many genes need to be simultaneously mutated to produce the same change in the structure. [126] defined such a rotation operator in the absolute encoding as, given a point at which the rotation is produced, all the remaining genes will be changed according to a mapping that depends on the angle to be rotated, i.e. to rotate 90 degrees clockwise $U \rightarrow R, R \rightarrow D, D \rightarrow L, L \rightarrow U$. In the current example, Fig. (4.7(a)) is encoded as $S_{abs} = DLLU\mathbf{RUULL}$ while the first mutated configuration (Fig. (4.7(b))) is $S^1_{abs} = DLLU\mathbf{ULLDD}$ and the rest configuration is $S_{abs} = DLLU\mathbf{LDDRR}$.

In our algorithm, if a gene is mutated, it will follow the regular: $L \rightarrow$

$R, R \rightarrow F, F \rightarrow L$.

### 4.5.3 Fitness function

#### 4.5.3.1 Computing number of H-H Contacts

In the conventional way of encoding in GA approaches ( [126, 163, 197]), the first and second amino acids in the sequence have fixed position on the square lattice on which the first one is supposed to be at the origin without loosing the generality. Thus, given a sequence of $N + 2$ amino acids, it needs to encode each chromosome as a sequence of only $N$ genes, each gene receives any value in $\{L, R, F\}$. Given such a chromosome and knowing the positions of the first and second residues (amino acid), one can calculate the number of H-H contacts and the number of collision positions (each of which has more than 1 residue being put at).

Search space of this approach has, however, contained at least 4 distinct solutions for each conformations (both feasible and infeasible one). It means that, given a certain conformation, there are at least 4 distinct chromosomes under the form of $\{L, R, F\}^N$ encoding that conformation ! Technically, if we assume the length between two adjacent residues on the lattice is 1, then the smallest-in-size square needed to cover any conformation will definitely has the size 2(N+1). This problem tends to be a kind of 'redundant' encoding.

In this report, we proposed a new way of encoding that overcomes this issue. From the fact that given a sequence $\{L, R, F\}^N$ there always exists a square of size $N + 1$ that covers any conformation induced by Relative or Absolute Encoding, hence we can assume there is a fixed square of size $N + 1$ in prior and put the conformation into that square without changing its "structure"

(actually, this can be done by using 'shifting and/or rotating' operators on the previous position of the conformation). Moreover, to do this, instead of fixing positions of the first and second residues, one must fix positions of the middle residue and its adjacent residue. Also, it is a bit technical that when one draws position of the whole conformation on square lattice, one needs to start from the residues staying on the same side in relative to the middle (right side, for instance), then from the other side (left). After that, the number of H-H contacts and collision positions are found by the same way as in previous approaches.

Clearly, using this proposed technique overcomes such a redundant issue in encoding and may affect the convergent speed positively.

**4.5.3.1.1 Choosing fitness function** It is well-known that this is always one of key steps in designing Genetic Algorithms. If the fitness function is badly designed, it would guide wrongly the search progress.

Our designed GA for the PSP allows infeasible conformations to exist, thus its fitness function must take into account the number of collision positions. Since there have been no suggestion in details of how to design such a fitness function with the existence of infeasible conformations found in the works already mentioned, we tried to list standards (conditions) on which a fitness function should be built.

Let $x$ and $y$ denote the number of H-H contacts and the number of collision positions, respectively, and let the fitness function be $F(x, y)$. Below are standards for building $F(.,.)$:

1. $F(x, y) > 0 \; \forall x, y \in N$ (a must).

2. $F(x, y_1) > F(x, y_2) \Longleftrightarrow y_1 < y_2$ (a must).

3. $F(x_1, y) > F(x_2, y) \Longleftrightarrow x_1 > x_2$ (a must).

4. $F(x_1, 0) > F(x_2, y)\ \forall y > 0$ (optional).

The standards (1,2,3) are compulsory, while the fourth needs to be paid more attention to. Indeed, if we want to make sure that the *best infeasible* conformation has a fitness lower than the *worst feasible* conformation, then the fourth should be used. But, applying this standard can lead to a biased search process in which it ignores *potential* conformations (have $y > 0$ but need few changes - mutation steps - to become feasible ones with x value much higher).

In addition, with the same reason as for the standard 4, knowing clearly the relationship between $F(x_1, y_1)$ and $F(x_2, y_2)$ if $x_1 < x_2,\ y_1 < y_2$ should also be taken care of.

#### 4.5.3.2 Results

In this section, performance of the approach using our proposed technique (see section 4.5.3.1) is compared with that not using this technique (conventional way).

Below are the benchmark instances which were used at least in [126, 163, 197] for the purpose of comparing performance. We will pick some of them for our tests.

**1. (20)** HPHPPHHPHPPHPHHPPHPH;

**2. (24)** HHPPHPPHPPHPPHPPHPPHPPHH;

**3. (25)** PPHPPHHPPPPHHPPPPHHPPPPHH;

**4. (36)** PPPHHPPHHPPPPPHHHHHHHHPPHHPPPPHHPPHPP;

**5. (48)** PPHPPHHPPHHPPPPPHHHHHHHHHHPPPPPPHHPPHHPPHPPH HHHH;

**6. (50)** HHPHPHPHPHHHHPHPPPHPPPHPPPPHPPPHPPPHPHHHHPHP HPHPHH;

**7. (60)** PPHHHPHHHHHHHHPPPHHHHHHHHHHPHPPPHHHHHHHHHHHH PPPPHHHHHHPHHPHP;

**8. (64)** HHHHHHHHHHHHPHPHPPHHPPHHPPHPPHHPPHHPPHPPHHPP HHPPHPHPHHHHHHHHHHHH;

**9. (20)** HHHPPHPHPHPPHPHPHPPH;

Table 4.3: Found results for these sequences in literature. The bolded values of $E^*$ are the surely minimum energies for the given protein sequences, the other values have been the best known so far.

| Instances | | |
|---|---|---|
| Seq. No. | Length | $E^*$ |
| 1 | 20 | **-9** |
| 2 | 24 | **-9** |
| 3 | 25 | **-8** |
| 4 | 36 | -14 |
| 5 | 48 | -23 |
| 6 | 50 | -21 |
| 7 | 60 | -36 |
| 8 | 64 | -42 |
| 9 | 20 | **-10** |

With the purpose of fairly comparing our proposed technique (mentioned in section 4.5.3.1) with others, the code for both approaches was written on Matlab 6.5 with a slight difference in the part of calculating fitness, and run on the same computer which is a Pen IV, 2.6Ghz, 512MRAM.

**4.5.3.2.1 Algorithm Settings** In all tests, common settings are below.

- .95 Crossover probability.
- 1-point crossover, roulette-wheel selection[30].
- Mutation probability less than .005.
- Fixed population size at 100.
- Randomly initial population.
- Non-valid structures allowed by penalized counting the number of positions in the grid where collisions take place (see section 4.5.3.1.1)
- Complete population replacement (A) or the mating population and the current one are ranked then choosing the best ones (B). In both scheme just an elite individual is passed to the next generation.
- Two parents are mated with a probability $P_X$ to form two offsprings. If the mating does not happen then the both parent will be considered as the two offsprings.

**4.5.3.2.2 Empirical results** Listed below are specific fitness functions we will consider for both GA approaches. From now on, we call the GA approach with our proposed technique $GA_{reduced}$, and the other (not using this technique) $GA_{not-reduced}$.

[30]As to fairly compare the performance of GAs on different ways of representing conformation, selection operators must be the same in GA implementations. Therefore, we use wheel roulette selection operator for this study although tournament selection is more the state-of-the-art.

1. $F(x, y) = ((x + 1)^2 * log(2 + x^5))/(2 + y)^2$;

2. $F(x, y) = ((x^{0.7} + 1) * log(2 + x^2))/((y * (1 + log(2 + y)) + 1/log(2)) * log(2 + y))$;

3. $F(x, y) = (x + 1)^2/(y + 1)$;

4. $F(x, y) = (N + 2 + x - y)/(y + 1)$;

5. $F(x, y) = (x^{0.5} * log(2 + x^5) + 0.01)/(y^{0.85} + 1.0)$;

Table 4.4: Results with different fitness functions for both GA approaches on **sequence 1**. *sq* is the best solution quality over all runs, $n_{opt}$ is the number of runs the algorithm finds *sq*, $n_{runs}$ is the total number of runs, *% suc.* is the percentage of runs in which solution quality *sq* was achieved. There are 100 generations for each trial and scheme (A) of the population replacement is used. $GA_{reduced}$ is GA using our proposed technique.

| Fit. function | $GA_{reduced}$ | | | $GA_{not-reduced}$ | | |
|---|---|---|---|---|---|---|
| No. | *sq* | $n_{opt}/n_{runs}$ | *% suc.* | *sq* | $n_{opt}/n_{runs}$ | *% suc.* |
| 1 | -8 | 2/10 | 20.0 | -8 | 1/10 | 10.0 |
| 2 | -8 | 6/10 | 60.0 | -8 | 5/10 | 50.0 |
| 3 | -8 | 6/10 | 60.0 | -8 | 4/10 | 40.0 |
| 4 | -8 | 2/10 | 20.0 | -7 | 1/10 | 10.0 |
| 5 | -8 | 5/10 | 50.0 | -8 | 3/10 | 30.0 |

From Table(4.4, 4.5), the fitness functions 2,3,5 seem to be most suitable for both $GA_{reduced}$ and $GA_{not-reduced}$.

Furthermore, as can be seen from these two Tables, with the same settings $GA_{reduced}$ always outperforms $GA_{not-reduced}$ in terms of the number of times it hits the best solution (quantity $n_{opt}/n_{runs}$).

From Table (4.6), $GA_{reduced}$ gave a faster convergence to current best solutions than $GA_{not-reduced}$ did. A possible reason is because of the search space

Table 4.5: Results with different fitness functions for both GA approaches on **sequence 4**. *sq* is the best solution quality over all runs, $n_{opt}$ is the number of runs the algorithm finds *sq*, $n_{runs}$ is the total number of runs, *% suc.* is the percentage of runs in which solution quality *sq* was achieved. There are 150 generations for each trial and scheme (A) of the population replacement is used. $GA_{reduced}$ is GA using our proposed technique.

| Fit. function | $GA_{reduced}$ | | | $GA_{not-reduced}$ | | |
|---|---|---|---|---|---|---|
| No. | *sq* | $n_{opt}/n_{runs}$ | *% suc.* | *sq* | $n_{opt}/n_{runs}$ | *% suc.* |
| 1 | -12 | 2/10 | 20.0 | -12 | 1/10 | 10.0 |
| 2 | -13 | 2/10 | 20.0 | -13 | 1/10 | 10.0 |
| 3 | -13 | 1/10 | 10.0 | -13 | 1/10 | 10.0 |
| 4 | -12 | 1/10 | 10.0 | -12 | 1/10 | 10.0 |
| 5 | -12 | 3/10 | 30.0 | -12 | 1/10 | 10.0 |

Table 4.6: Compare performance between $GA_{reduced}$ and $GA_{not-reduced}$ The fitness function 2 is used in both implementations. There are 150 generations for sequences whose length is less than 40, and 300 generations for the rest.

| Instances | | | $GA_{reduced}$ | | | $GA_{not-reduced}$ | | |
|---|---|---|---|---|---|---|---|---|
| Seq. No. | Length | *E** | *sq* | $n_{opt}/n_{runs}$ | *% suc.* | *sq* | $n_{opt}/n_{runs}$ | *% suc.* |
| 1 | 20 | **-9** | -8 | 3/5 | 60.0 | -8 | 3/10 | 60.0 |
| 2 | 24 | **-9** | -8 | 3/5 | 60.0 | -8 | 5/10 | 50.0 |
| 3 | 25 | **-8** | -6 | 2/5 | 40.0 | -7 | 1/10 | 10.0 |
| 4 | 36 | -14 | -13 | 2/5 | 40.0 | -13 | 2/5 | 40.0 |
| 5 | 48 | -23 | -16 | 1/5 | 20.0 | -15 | 1/10 | 10.0 |
| 6 | 50 | -21 | -18 | 1/10 | 10.0 | -18 | 1/10 | 10.0 |
| 9 | 20 | **-10** | -6 | 5/5 | 100.0 | -7 | 1/10 | 10.0 |

of $GA_{reduced}$ is more compact than that of $GA_{not-reduced}$ while both of them are still containing all possible solutions. Any 2D conformation is modeled as more than two distinct points in search space of $GA_{not-reduced}$ which diverges

the search because the search process might focus on distinct regions around good distinct genes even though those distinct genes models the same solution (i.e. the same 2D conformation). Because of this potential divergence, $GA_{not-reduced}$ could converge to best-so-far solutions slower than $GA_{reduced}$ does.

### 4.5.4 Discussion

In this section, we have shown a sufficient condition on Euclidean Traveling Salesman Problem, when deploying the SDEB method. This condition guarantees all optimal solutions of a given instance (which meets the condition) will be in the search space formed by choosing a certain number of edges to form bridges. This condition is currently proven for the case that there are two subproblems. Probably because of some conditional constraints, this condition may not be seen frequently in real-world instances of ETSP, and consequently the application range of this result is quite restricted.

It would be interesting to find other sufficient conditions with the same purpose as this proven condition, but for higher number of subproblems. Moreover, some early results obtained from ongoing numerical experiments carried out to investigate the practical usefulness of results from Theorems (4.4.1, 4.4.2) were satisfactory (at least) to clustered Euclidean Traveling Salesman Problem instances, and are the main motivation to continue those ongoing experiments.

## 4.6 Summary

Empirical runtime analysis for the approach based on divide and conquer principle to large scale instances of optimization problems was presented in this chapter. The analysis has addressed scenarios to achieve better lower bound on speedup of any SDEB approach in relative to its PUND approach. Those scenarios include also the asymptotical case when size of input instance approaches to infinity. Result on the asymptotical case showed that the approach theoretically gain a faster runtime in comparing with its counterpart but there exists a finite limit (upper bound) for the gain of runtime reduction. And that finite limit can be estimated from the empirical complexity of an algorithm used to solve underlying problems and the number of parts (subproblems) going through the searching process by the SDEB algorithm. The estimation for that upper limit is a tight limit under given assumptions for both asymptotical and non-asymptotical cases.

Relationship between these findings with runtime analysis for SDEB methods that do not govern number of subproblems when solving the original instance, for examples POPMUSIC and Dantzig-Wolf Decomposition Principle, is addressed in this chapter. Results on the upper and lower bounds can be transformed to these methods when total size of subproblems they were solving is very close to size of the original instance.

In this chapter, we have also shown a sufficient condition on structure Euclidean Traveling Salesman Problem so as to obtain the most simple guarantee on optimality of found solutions of a SDEB method. This condition guarantees all optimal solutions of a given instance (which meets the condition) must be in the search space formed by choosing a certain number of edges

to form bridges. This condition is currently proven for the case that there are two subproblems. Probably because of some conditional constraints, this condition may not be seen frequently in real-world instances of ETSP, and consequently the application range of this result is quite restricted. This restriction on number of subproblems of two is removed in theorems (4.4.1) and (4.4.2). However, results achieved in these two theorem are imposed by the assumptions that perhaps suitable to large clustered ETSP.

It should be interesting to find other sufficient conditions similar to this, but for higher number of subproblems without imposing any assumption. Moreover, numerical experiments need to carry out to investigate the practical usefulness of results from theorem (4.4.1) and (4.4.2). We hope that these results probably give very fruitful solutions to (at least) clustered Euclidean Traveling Salesman Problem.

# Chapter 5

# Conclusions and Future Works

## 5.1 Summary of Contributions of the Thesis

This research investigates issues of solving combinatorial optimization problems using Ant Colony Optimization metaheuristic for small and medium scale instances and decomposition-based methods for large scale ones. The primary findings and contributions of this research are summarized as follows.

### 5.1.1 Ant Colony Optimization

The first part of the study on Ant Colony Optimization has investigated the role of the trade-off technique which is used popularly in Ant-based algorithms. We have approached this problem in two directions. One of them is to model the ACO metaheuristic with this trade-off mechanism by extending the model so-called Graph-based Ant System proposed by Gutjahr [106]. The direction is to carry out simulation-based experiments for which a dynamically linear updating rule for trade-off mechanism is applied.

Following the first direction, we conducted a theoretical study for a more generalized Graph-based Ant System framework into which the trade-off mechanism is incorporated. We have carried out the analysis of convergence properties of this generalized model as the value of the exploiting parameter is fixed over time. Results from the analysis show that all convergence properties of the original GBAS (without containing the trade-off mechanism) are

still intact in our generalized version of GBAS. This finding shows the theoretical soundness of using the trade-off mechanism in Ant-based implementations in terms of the convergence property.

In this part of the study we did not point out how much the incorporation of this mechanism into the model affects the convergence speed of Ant-based algorithms derived from this generalized model in comparison to those derived from original GBAS model. Nevertheless, this limitation is acceptable because the same obstacle has been also encountered in other studies due to the complexity of analyzing the convergence speed for ACO-based algorithms. There has been no study in the field which investigates the convergence speed of any Ant-based algorithm.

For the experimental direction, we introduced a time-dependent trade-off mechanism into ACO-based algorithms by dynamically updating the value of the exploiting parameter using a linear rule. ACO-based algorithms always either employ the trade-off mechanism with a fixed-over-time value of the exploiting parameter or exclude the mechanism from their implementations. The experimental results show that, for all Ant-based algorithms using the mechanism, with the presence of local search procedure, those with the mechanism incorporated have a better performance than those without using the mechanism. The good performance of this dynamical approach can be explained as follows. In the late stage of ACO-based algorithms, it is most likely that the search space guided by the pheromone matrix is narrowed down to the search areas including feasible solutions around the best-so-far solutions. Thus, gradually increasing or decreasing the value of the exploiting parameter over runtime, is likely to increase the probability of discovering distinct

solutions in other search areas of search space in the late stage. This can be considered as a promising method to prevent the stagnation from occurring in the late stage of ACO-based algorithms.

### 5.1.2 Decomposition-based Search Algorithms

This part of the study aimed at providing theoretical evidences to explain the time efficiency of decomposition-based over non-decomposition-based algorithms. By modeling the runtime functions of decomposition-based algorithms as under the polynomial-like form, we obtained the theoretical evidence that strongly supports experimental results-based claims about the faster runtime of decomposition-based algorithms in comparison to the non-decomposition-based ones for the same inputs. This evidence shows that there exist finite bounds on the speedup of the decomposition-based algorithms in relation to their non-decomposition-based counterparts. Moreover, these finite bounds are determined by the number of subproblems and the runtime function of the algorithm used to solve the underlying problem. This finding points out that the runtime efficiency of decomposition-based algorithms over non-decomposition-based ones can be measured quantitatively.

From experimental analysis conducted in other studies, decomposition-based algorithms are clearly superior to non-decomposition-based ones in large-scale instances. To understand this phenomenon, we carried out an asymptotical-runtime analysis which the input size approaches infinity. Interestingly, for this asymptotical case, the bounds on the ratio are still identifiable using features of the highest order component of the runtime function of the problem-solving algorithm (used in the decomposition-based method).

Not only do the findings from this theoretical study provide evidence for the experiment-based claim, but based on findings we also suggest some methods to increase the runtime efficiency of decomposition-based algorithms by setting the maximum size of subproblems at desired values.

Since the optimal solutions may be excluded from the search space of an inexact decomposition-based algorithm due to the decomposition and assembling process, the optimality of those methods' solutions is not completely guaranteed. In this part of study, we developed a simple decomposition-based algorithm solving Euclidean Traveling Salesman Problem (ETSP) in order to illustrate a new approach of studying the the issue of a decomposition-based algorithm in terms of the optimality of its solutions. The approach is to find out conditions for structure of optimization instances under which all optimal solutions of satisfied instances are guaranteed to appear in the search space of a given inexact decomposition-based method. Given a simple decomposition-based algorithm for ETSP, we proved a sufficient condition on the structure of ETSP's instances such that all optimal solutions are guaranteed to stay in the search space of the algorithm. This sufficient condition has been found for the case of two subproblems. We admit that studying such conditions is not easy even for a simple decomposition-based method which we established a result for the case of two subproblems. Indeed, when the number of subproblems is more than two, it would be very challenging to use the same or a bit modified approach to solve for that case of, for instance, three subproblems.

## 5.2 Future Works

Turning to suggestions for future research, several problems remain open for both theoretical and experimental study. Most notably, incorporating the ..

### 5.2.1 Ant Colony Optimization

We have studied an extended model of GBAS which inherits limitations due to strong constraints (on required structure of optimization problems and the method of encoding solutions) of GBAS. To overcome those inherited limitations, Gutjahr developed a variant of GBAS whose systematic parameters are time-dependent [107]. So, we can adopt a similar way to relax those strong constraints in the extended model. Moreover, another coherent limitation of our extended model is to keep the value of the exploiting parameter constant over time. For a recommendation of further study, a time-dependent version of this extended model which can relax those strong constraints and adjust the parameter over time should be carried out. To strengthen the conclusion of that the convergence speed of EGBAS model might be controlled not only by systematic parameters but also by the exploiting parameter, more extensive experiments should be done.

There are still two remaining open interesting questions related to study ACO-based algorithms' performance. One of them is to find out how convergence speed of Ant-based algorithms functions in relationship with system parameters or problems being tackled. The other is on examining whether or not performance of these algorithm is improved if the Markov property of ACO framework is violated. If the answer to the last question is posi-

tive, i.e. the performance of ACO is improved, then obviously, an efficient pseudo-probabilistic method which is similar to ACO can be derived from that study.

Building on the idea of employing the strategy of varying-over-time population size from Evolutionary Computation, future works should investigate the performance of ACO-based algorithms in the case that number of agents changes over time. To the best of our knowledge, no one has conducted any work on this idea in ACO although results in the work by Gutjahr [106] provided a rough indication of how the idea works.

Finally, for the approach of tuning values of a group of systematic parameters, in that experimental study, we did not examine any non-linear rule for dynamically tuning the exploiting parameter. Neither did we study any linear or non-linear rule for updating a set of parameters. However, with the simple rule for a specific parameter, the performance of studied Ant-based algorithms has been slightly improved. Thus, the finding of this approach suggests that a rule for simultaneously dynamically updating values of "representative" parameters would improve performance of Ant-based algorithms. Representative parameters could be chosen in a hierarchical way such that the number of such parameters should be reduced as many as possible while increasing influence of those parameters on performance of the algorithms. Therefore , we suggest for a further study on non-linear rules for a certain parameter or combining linear with non-linear rules but for a group of systematic parameters.

### 5.2.2 Decomposition-based Algorithms

By modeling runtime functions of the problem-solving algorithms - which are used in the conquering stage of the decomposition-based methods as functions of size of input instances of optimization problems, we showed the efficiency in runtime of decomposition-based methods over the corresponding non-decomposition-based. The efficiency is obvious for the asymptotical case when the size of input instances is very large. However, for non-asymptotical case when the size of input instances can receive any value, the efficiency is showed for algorithms whose runtime functions are polynomial-like with positive coefficients. The constraint on the sign of coefficients of those functions has restricted the application extent of the model.

We recommend a further study on an extended model in which that constraint is relaxed, or more specifically the class of runtime functions represented in that extended model should be extended to that which contains not only all polynomial-like functions with real-valued coefficients[1] but also logarithm-like or compound functions. We predict that findings of this recommended study for such extended models may introduce relative values of certain metrics on problem structures with which if problem instances are satisfied then using decomposition-based methods to solve those instances is more suitable than it is when those instances are not satisfied.

With the new approach in resolving the issue related to the optimality of solutions of inexact decomposition-based algorithms, there is enough room for further theoretical studies about the sufficient conditions on the structure of problem instances to improve quality of solutions produced by those al-

[1]These coefficients can receive positive or negative values.

gorithms. Those sufficient conditions are strongly dependent on the type of problems under considerations or the specific (inexact) decomposition-based algorithms employed.

# Appendix A

# Author's Publications


1. H. T. Dinh and A. A. Mamun. The speedup of the cluster-based approach in the divide and conquer paradigm. In *Proceedings of 4th EU-ME Workshop on Design and Evaluation of Advanced Hybrid Meta- Heuristics*, 2004.

2. H. T. Dinh and A. Al-Mamun. A combination of clustering algorithm with ant colony optimization for large clustered traveling salesman problem. *WSEAS Transactions on Systems*, 3(3):1221-1227, 2004.

3. H. T. Dinh, A. A. Mamun, and D. Hieu. Dynamically Updating the Exploiting Parameter in Improving Performance of Ant-Based Algorithms. In N. Megiddo, Y. Xu, and B. Zhu, editors, *Proceedings of Algorithmic Applications in Management, Xian, China*, volume 3521 of *Lecture Notes in Computer Science*, pages 340-349. Springer, 2005.

4. H. T. Dinh, A. A. Mamun, and H. T. Huynh. A generalized version of graph-based ant system and its applicability and convergence. In *Proceeding of 4th IEEE International Workshop on Soft Computing as Transdisplinary Science and Technology (WSTST'05)*, pages 949-958. Springer-Verlag, 2005.

5. H. T. Dinh, D. Hieu, and A. A. Mamun. On designing clustering and combining algorithms in POPMUSIC and 3-stage method. In *7th International Conference on Artificial Evolution EA2005, Lille, France*, CDROM, 2005.

6. H. T. Dinh, A. A. Mamun, and H. T. Huynh. On extension of Graph-based Ant System with a tradeoff mechanism and its applicability and convergence. *Submitted to Journal of Heuristics*, Sep 2007.

# Appendix B

# Proof of Theorem (3.2.2)

We have

$$P_m = Pr\left\{E_m^{(1)}\right\} = Pr\left\{E_m^{(2)}\right\} = .. = Pr\left\{E_m^{(s)}\right\}.$$

By Corollary 3.2.5 and Lemma (3.2.4), because the event $F^*$ can be presented as

$$F^* = \neg(B_1 \wedge B_2 \wedge \ldots),$$

we obtain

$$Pr\{F^*\} = 1 - \lim_{m\to\infty} Pr\{B_1 \wedge \ldots \wedge B_m\} \geq 1 - \lim_{m\to\infty}\left[\prod_{i=1}^{m}(1 - c^{i-1}p)\right]^S = 1 - w(p, c, S).$$

Since $w(p, c, S) \leq exp\left(-\frac{Sp}{1-c}\right)$, so

$$\mathbf{Pr\{F^*\} \geq 1 - \epsilon/4}$$

**by choosing appropriate values for S or** $\rho$(*). In addition, due to

$$1 > Pr\{F^*\} = Pr\{F_1 \vee F_2 \vee \ldots\} = \sum_{m=1}^{\infty} Pr\{F_m\}.$$

Then there is an integer $K = K(\epsilon)$ such that

$$\sum_{m=K+1}^{\infty} Pr\{F_m\} < \frac{\epsilon}{4}.$$

Let $F = F_1 \vee \ldots \vee F_K$, hence

$$Pr\{F\} = \sum_{m=1}^{K} Pr\{F_m\} \geq Pr\{F^*\} - \frac{\epsilon}{4} \geq 1 - \frac{\epsilon}{2}.$$

By Lemma (C.0.6),

$$Pr\left\{E_{m'}^{(s)}|F_m\right\} \geq 1 - \frac{\epsilon}{2}, \tag{B.0.1}$$

for all $m' \geq m + d''''(\epsilon/2, m)$. Let

$$d(\epsilon) = \max\left(d''''(\frac{\epsilon}{2}, 1), \ldots, d''''(\frac{\epsilon}{2}, K)\right),$$

and

$$m_0 = m_0(\epsilon) = K + d(\epsilon)$$

. Then for $m \leq K$, (B.0.1) holds for all $m' \geq m_0$

$$\begin{aligned} P_{m'} &= Pr\left(E^{(1)}_{m'}\right) = Pr\left(E^{(1)}_{m'} \wedge (\neg F \vee F)\right) \\ &= Pr\left(E^{(1)}_{m'}|F_1\right) Pr(F_1) + \ldots + Pr\left(E^{(K)}_{m'}|F_K\right) Pr(F_K) \\ &+ Pr\left(E^{(1)}_{m'}|\neg(F_1 \vee \ldots \vee F_K)\right) \cdot Pr(\neg(F_1 \vee \ldots \vee F_K)) \qquad \text{(B.0.2)} \\ &\geq Pr\left(E^{(1)}_{m'}|F_1\right) Pr(F_1) + \ldots + Pr\left(E^{(K)}_{m'}|F_K\right) Pr(F_K) \\ &\geq \left(1 - \frac{\epsilon}{2}\right)(Pr(F_1) + \ldots + Pr(F_K)) \geq \left(1 - \frac{\epsilon}{2}\right)\left(1 - \frac{\epsilon}{2}\right) \geq 1 - \epsilon. \end{aligned}$$

From the remark (*) and (B.0.2), theorem (3.2.2) is successfully proven.

# Appendix C

# Restatement of lemmas and corollaries used to prove GBAS's convergence

**Lemma C.0.1.** *[Lemma 4.1 in [106]] The probability $Pr(\neg B_m)$ which at least one agent traverses the optimal walk in cycle m is not less than $1-(1-c^{m-1}p)^S$, where $c=(1-\rho)^L$ and $p=\gamma^L\cdot\prod_{(k,l)\in w^*}\tau_{kl}(1)$ with $\gamma$ defined by ( 3.2.11 ).*

**Corollary C.0.2.** *[Corollary 4.1 in [106]] The conditional probability $Pr(\neg B_m|B_1\wedge\ldots\wedge B_{m-1})$ that at least one agent traverses in cycle m the optimal walk, given that all agents have never traversed the optimal walk in all previous cycles, is not less than $1-(1-c^{m-1}p)^S$, where c is computed as in Lemma (C.0.1).*

**Lemma C.0.3.** *[Lemma 4.2 in [106]] For each $\epsilon>0$ and each $m\in N$ there is an integer $d(\epsilon,m)\in N$ such that $\forall\ m'\geq m+d(\epsilon,m)$*

$$\begin{cases} Pr\left\{\left|\tau_{kl}(m')-\dfrac{1}{L}\right|<\epsilon,\ \forall(k,l)\in w^*|F_m\right\} & \geq 1-\epsilon \\ \\ Pr\{\tau_{kl}(m')<L\epsilon,\ \forall(k,l)\notin w^*|F_m\} & \geq 1-\epsilon \end{cases} \tag{C.0.1}$$

**Lemma C.0.4.** *[Lemma 4.3 in [106]] Let $u^*(k)$ indicates the partial walks on $w^*$ leading to node k ($k\in w^*$). Then for each $\epsilon>0$ and each $m\in N$ there is an integer $d''(\epsilon,m)\in N$ such that $\forall(k,\ l)\in w^*$ and $\forall m'\geq m+d''(\epsilon,m)$*

$$Pr\{p_{kl}(m',u^*(k))\geq 1-\epsilon|F_m\}\geq 1-\epsilon.$$

**Corollary C.0.5.** *[Corollary 4.2 in [106]] With the notation in Lemma (C.0.4), let*

$$Y_{m'}=\prod_{(k,l)\in w^*}p_{kl}(m',u^*(k)). \tag{C.0.2}$$

*Then, for each* $\epsilon > 0$ *and each* $m \in N$*, there is an integer* $d'''(\epsilon, m) \in N$ *such that*

$$Pr\{Y_{m'} \geq 1 - \epsilon | F_m\} \geq 1 - \epsilon.$$

*for all* $m' \geq m + d'''(\epsilon, m)$*.*

**Lemma C.0.6.** *[Lemma* 4.4 *in [106]] For each* $\epsilon > 0$ *there is an integer* $d''''(\epsilon, m) \in N$*, such that for a fixed agent* $s$ *and for all* $m' \geq m + d''''(\epsilon, m)$

$$Pr\left\{E_{m'}^{(s)} | F_m\right\} \geq 1 - \epsilon.$$

# Appendix D

# List of Abbreviations

| | |
|---|---|
| ACO | Ant Colony Optimization |
| ACS | Ant Colony System |
| AS | Ant System |
| BWAS | Best-Worst Ant System |
| COP | Combinatorial Optimization Problem |
| DC | Divide and Conquer |
| EC | Evolutionary Computation |
| EDA | Estimation of Distribution Algorithm |
| EGBAS | Extension of Graph-based Ant System |
| EP | Evolutionary Programming |
| ES | Evolutionary Strategies |
| ETSP | Euclidean Traveling Salesman Problem |
| GA | Genetic Algorithm |
| GBAS | Graph-based Ant System |
| GRASP | Greedy Randomized Adaptive Search Procedure |
| I&D | Intensification and Diversification |
| LP | Linear Programming |
| MA | Memetic Algorithm |
| MMAS | Max-Min Ant System |
| NFL | No Free Lunch |
| POPMUSIC | Partial Optimization Metaheuristic Under Special Intensification Conditions |
| PUND | Purely Non-Decomposition-based Method |
| SA | Simulated Annealing |
| SDEB | Serial Decomposition-based Method |
| TS | Tabu Search |
| TSP | Traveling Salesman Problem |
| VND | Variable Neighborhood Descent |
| VNS | Variable Neighborhood Search |

# BIBLIOGRAPHY

[1] E. Aarts and J. Korst. *Simulated annealing and Boltzmann machines: A stochastic approach to combinatorial optimization and neural computing.* John Wiley and Sons, Inc.: New York, 1990.

[2] A. Aho, J. Hopcroft, and J. Ullman. *The Design and Analysis of Computer Algorithms.* The MIT Press, 1974.

[3] G. Amdahl. Validity of the single-processor approach to achieving large scale computing capabilities. In *AFIPS Conference Proceedings*, volume 30 (Atlantic City, N.J., Apr. 18-20), pages 483–485. AFIPS Press, Reston, Va., 1967.

[4] D. Applegate, W. Cook, and A. Rohe. Chained lin-kernighan for large traveling salesman problems. *INFORMS Journal on Computing*, 15(1):82–92, 2003.

[5] S. Arora. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *Journal of the ACM (JACM)*, 45(5):753–782, 1977.

[6] S. Ashour. A decomposition approach for the machine scheduling problem. *International Journal of Production Research.*, 6:109–122, 1967.

[7] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation: Combinatorial optimization problems and their approximability properties.* Springer Verlag, 1999.

[8] T. Bäck, D. B. Fogel, and Z. Machalewics, editors. *Handbook of Evolutionary Computation.* Institute of Physics Publishing Ltd., Bristol, UK, 1997.

[9] S. Baluja. Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Technical Report CMU-CS-94-163, Carnegie Mellon University,, Pittsburgh, PA, 1994.

[10] S. Baluja and R. Caruana. Removing the genetics from the standard genetic algorithm. In A. Prieditis and S. Rusell, editors, *Machine Learning: Proceedings of the 12th International Conference.*, pages 38–46. Morgan Kaufmann Publishers, 1995.

[11] J. W. Barnes, B. Dimova, S. P. Dokov, and A. Solomon. The theory of elementary landscapes. *Applied Mathematics Letters.*, 16(3):337–343, 2003.

[12] J. W. Barnes, V. D. Wiley, J. T. Moore, and D. M. Ryer. Solving the aerial fleet refueling problem using group theoretic tabu search. *Mathematical and Computer Modeling.*, 39:617–640, 2004.

[13] M. Bastos and C. Ribeiro. Reactive tabu search with path-relinking for the steiner problem in graphs. In *Proceedings of the Third Metaheuristics International Conference*, pages 31–36, 1999.

[14] R. Battiti. Reactive search: Toward self-tuning heuristics. In V. J. Rayward-Smith, I. H. Osman, C. R. Reeves, and G. D. Smith, editors, *Modern Heuristic Search Methods.*, pages 61–83. John Wiley & Sons, Chichester, UK, 1996.

[15] E. M. L. Beale. *Mathematical Progamming in Practice.* John Wiley & Sons, Inc., New York, 1968.

[16] J. Berger, M. Sassi, and M. Salois. A hybrid genetic algorithm for the vehicle routing problem with time windows and itinerary constraints. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-1999)*, pages 44–51. Morgan Kaufmann Publishers, San Fransisco, CA, 1999.

[17] M. Birattari, editor. *The Problem of Tuning Metaheuristics as Seen from a Machine Learning Perspective*. DISKI 292, Infix/Aka, Berlin, Germany, 2005.

[18] M. Birattari, M. Zlochin, , and M. Dorigo. Towards a theory of practice in metaheuristics design: A machine learning perspective. *Theoretical Informatics and Applications.*, 40(2):353–369, 2006.

[19] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35:268–308, 2003.

[20] C. Blum, A. Roli, and E. Alba. An introduction to metaheuristic techniques. In E. Alba, editor, *Parallel Metaheuristics: A New Class of Algorithms*. John Wiley & Sons, Inc., 2005.

[21] J. S. D. Bonet, C. L. Isbell, and P. Viola. MIMIC: Finding optima by estimating probability densities. In M. J. M. Mozer and T. Petsche., editors, *Advances in Neural Information Processing Systems, volume 9.*, pages 424–431. Morgan Kaufmann, Amherst, Massachusetts, 1997.

[22] Y. Borenstein and R. Poli. No free lunch, kolmogorov complexity and the information landscape. In *The 2005 IEEE Congress on Evolutionary Computation.*, pages 2784–2791 Vol. 3, 2005.

[23] O. Bräysy. A reactive variable neighborhood search for the vehicle-routing problem with time windows. *INFORMS Journal on Computing*, 15(4):347–368, 2003.

[24] E. K. Burke, J. P. Newall, and R. F. Weare. A memetic algorithm for university exam timetabling. In *1st International Conference on the Practice and Theory of Automated Timetabling (ICPTAT'95, Napier University, Edinburgh, UK, 30th Aug - 1st Sept 1995)*, pages 496–503, 1995.

[25] R. H. Byrd, R. B. Schnabel, and G. A. Shultz. A trust region algorithm for nonlinearly constrained optimization. *SIAM Journal on Numerical Analysis*, 24(5):1152–1170, 1987.

[26] G. D. Caro and M. Dorigo. Antnet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research*, 9:317–365, 1998.

[27] L. Cavique, C. Rego, and I. Themido. A scatter search algorithm for the maximum clique problem. In *Essays and Surveys in Metaheuristics*, pages 227–244. Kluwer Academic Publishers, 2001.

[28] M. Celis, J. E. Dennis, and R. A. Tapia. A trust region strategy for nonlinear equality constrained optimization. In P. Boggs, R. Byrd, and R. Schnabel, editors, *Numerical Optimization, Philadelphia: SIAM.*, pages 71–82, 1985.

[29] N. Christofides, A. Mingozzi, and P. Toth. The vehicle routing problem. In N. Christofides, A. Mingozzi, P. Toth, and C. Sandi, editors, *Combinatorial Optimization*, pages 315–338. Wiley, Chichester, 1979.

[30] V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research.*, 4(3):233–235, 1979.

[31] B. Codenotti and L. Margara. Local properties of some np-complete problems. Technical Report. TR-92-021, International Computer Science Institute, University of California at Berkeley., 1992.

[32] C. A. C. Coello. An updated survey of GA-based multiopjective optimization techniques. *ACM Computing Surveys.*, 32(2):109–143, 2000.

[33] C. A. C. Coello. Discovery of RNA structural elements using evolutionary computation. *Nucleic Acids Research.*, 30(23):5310–5317, 2002.

[34] B. Colletti and J. Barnes. Group theory and metaheuristic neighborhoods. Technical Report. Series 99-02, Graduate Program in Operations Research, the University of Texas at Austin., 1999.

[35] B. Colletti and J. Barnes. Local search structure in the symmetric traveling salesperson problem under a general class of rearrangement neighborhoods. *Applied Mathematics Letters.*, 14:105–108, 2001.

[36] B. Colletti, J. Barnes, and D. Neuway. Using group theory to study transition matrices of metaheuristic neighborhoods. Technical Report. Series 2000-03, Graduate Program in Operations Research, the University of Texas at Austin., 2000.

[37] B. Colletti and J. W. Barnes. Using group theory to construct and characterize metaheuristic search neighborhoods. In C. Rego and B. Alidaee, editors, *Adaptive Memory and Evolution: Tabu Search and Scatter Search*, 2004.

[38] B. W. Colletti. *Group Theory and Metaheuristics*. PhD thesis, The University of Texas at Austin, 1999.

[39] A. Colorni, M. Dorigo, and V. Maniezzo. Distributed optimization by ant colonies. In F.Varela and P.Bourgine, editors, *Proceedings of the First European Conference on Artificial Life.*, pages 134–142. Elsevier Publishing, Amsterdam, 1991.

[40] A. R. Conn, N. I. M. Gould, and P. L. Toint. *Trust-region methods*. Society for Industrial and Applied Mathematics, 2000.

[41] D. T. Connolly. An improved annealing scheme for the QAP. *European Journal of Operational Reseach*, 9(3):93–100, 1990.

[42] J. Cordeau and G. Laporte. Tabu search heuristics for the vehicle routing problem. GERAD Technical Report G-2002-15, University of Montreal, Canada, 2002.

[43] O. Cordón, I. F. de Viana, and F. Herrera. Analysis of the best worst ant system and its variants on the *qap*. In M. Dorigo, G. D. Caro, and M. Sampels, editors, *Ant Algorithms*, volume 2463 of *Lecture Notes in Computer Science*, pages 228–234. Springer Verlag, Berlin, Germany., 2002.

[44] O. Cordón, I. F. de Viana, F. Herrera, and L. Moreno. A new ACO model integrating evolutionary computation concepts: The best-worst ant system. In M. Dorigo, M. Middendorf, , and T. Stützle, editors, *Abstract Proceedings of ANTS* 2000 - *From Ant Colonies to Artificial Ants: A Series of International Workshops on Ant Algorithms.*, pages 22–29. Universit Libre de Bruxelles, Belgium, 2000.

[45] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms.* The MIT Press, 2nd edition, 2002.

[46] J. R. Crino, J. T. Moore, J. W. Barnes, and W. P. Nanry. Solving the military theater distribution vehicle routing and scheduling problem using group theoretic tabu search. *Mathematical and Computer Modeling.*, 39:599–616, 2004.

[47] V. Cung, T. Mautor, P. Michelon, and A. Tavares. A scatter search based approach for the quadratic assignment problem. In T. Baeck, Z. Michalewicz, and X. Yao, editors, *Proceedings of IEEE International Conference on Evolutionary Computation and Evolutionary Programming, Indianapolis, United States of America.*, pages 165–170. IEEE Press, 1997.

[48] V. Cutello and G. Nicosia. An immunological approach to combinatorial optimization problems. volume 2527 of *Lecture Notes in Computer Science*, pages 361–370. Springer-Verlag, 2002.

[49] V. Cutello, G. Nicosia, M. Pavone, and J. Timmis. An immune algorithm for protein structure prediction on lattice models. *IEEE Transactions on Evolutionary Computation*, 11(1):101–117, 2007.

[50] G. Dantzig and P.Wolfe. Decomposition principle for linear programs. *Operations Research*, 8:101–11, 1960.

[51] A. Das and B. K. Chakrabarti. *Quantum Annealing and Related Optimization Methods*. Lecture Note in Physics, Vol. 679, Springer, 2005.

[52] R. Dawkins, editor. *The Selfish Gene*. Oxford University Press, 2006.

[53] M. Dell'Amico, A. Lodi, and F. Maffioli. Solution of the cumulative assignment problem with a well-structured tabu search method. *Journal of Heuristics*, 5:123–143, 1999.

[54] J. L. Deneubourg, S. Aron, S. Goss, and J. M. Pasteels. The self-organizing exploratory pattern of argentine ant. *Journal of Insect Behaviour.*, 3:159–168, 1990.

[55] K. Dill. Theory for the folding and stability of globular-proteins. *Biochemistry*, 24(6):1501–1509, 1995.

[56] H. T. Dinh, H. T. Dinh, and A. A. Mamun. On designing clustering and combining algorithms in POPMUSIC and 3-stage method. In *7th International Conference on Artificial Evolution (EA2005), Lille, France, CDROM*, 2005.

[57] H. T. Dinh and A. A. Mamun. The speedup of the cluster-based approach in the divide and conquer paradigm. In *Proceedings of 4th EU-ME Workshop on Design and Evaluation of Advanced Hybrid Meta-Heuristics.*, 2004.

[58] H. T. Dinh, A. A. Mamun, and H. T. Dinh. Dynamically Updating the Exploiting Parameter in Improving Performance of Ant-Based Algorithms. In N. Megiddo, Y. Xu, and B. Zhu, editors, *Proceedings of Algorithmic Applications in Management, Xian, China*, volume 3521 of *Lecture Notes in Computer Science*, pages 340–349. Springer, 2005.

[59] H. T. Dinh, A. A. Mamun, and H. T. Huynh. A generalized version of graph-based ant system and its applicability and convergence. In *Proceeding of 4th IEEE International Workshop on Soft Computing as Transdisplinary Science and Technology* (*WSTST'*05), pages 949–958. Springer-Verlag, 2005.

[60] B. Doerr, F. Neumann, D. Sudholt, and C. Witt. On the runtime analysis of the 1-ANT ACO algorithm. In *GECCO'07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 33–40. ACM, New York, NY, USA., 2007.

[61] M. Dorigo. *Optimization, Learning and Natural Algorithms (in Italian).* PhD thesis, PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1992.

[62] M. Dorigo and C. Blum. Ant colony optimization theory: A survey. *Theoretical Computer Science*, 344(2-3):243–278, 2005.

[63] M. Dorigo and G. D. Caro. The ant colony optimization metaheuristic. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas In Optimization*. McGraw-Hill, 1999.

[64] M. Dorigo, G. D. Caro, and L. Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 5:137–172, 1999.

[65] M. Dorigo and L. Gambardella. Ant colony system: A cooperative learning approach to the travelling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1:53–66, 1997.

[66] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on System, Man, and Cybernetics*, 26(1):28–41, 1996.

[67] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE Transaction on Systems, Man, and Cybernetics*, 26:29–41, 1996.

[68] M. Dorigo and T. Stützle. *http://www.metaheuristics.net*, 2000. Visited 2004.

[69] M. Dorigo and T. Stützle. *Ant Colony Optimization.* MIT Press, Cambridge, MA, 2004.

[70] R. O. Duda, P. E. Hart, and D. G. Stork, editors. *Pattern Classification.* Wiley-Interscience, 2nd edition, 2000.

[71] J. Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics.*, 17:449–467, 1965.

[72] A. Engelbrecht. *Computational Intelligence : An Introduction.* Hoboken, N.J. : J. Wiley and Sons, 2002.

[73] T. Feo and M. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization.*, 6:109–133, 1995.

[74] P. Festa and M. Resende. Grasp: An annotated bibliography. In C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics.*, pages 325–367. Kluwer Academic Publishers, 2002.

[75] M. Fleischer. Simulated annealing: past, present and future. In C. Alexopoulos, K. Kang, W. R. Lilegdon, and G. Goldsman, editors, *Proceedings of the 1995 Winter Simulation Conference.*, pages 155–161, 1995.

[76] C. Fleurent, F. Glover, P. Michelon, and Z. Valli. A scatter search approach for unconstrained continuous optimization. In *Proceedings of IEEE International Conference on Evolutionary Computation, Nagoya, Japan.*, pages 643–648, 1996.

[77] L. J. Fogel. Toward inductive inference automata. In *Proceedings of the International Federation for Information Processing Congress.*, pages 395–399, 1962.

[78] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution.* Wiley, 1966.

[79] L. Gambardella and M. Dorigo. Ant-Q: A reinforcement learning approach to the traveling salesman problem. In *International Conference on Machine Learning*, pages 252–260, 1995.

[80] L. Gambardella and M. Dorigo. Solving symmetric and asymmetric TSPs by ant colonies. In *IEEE Conference on Evolutionary Computation (ICE'96).* IEEE Press, 1996.

[81] L. Gambardella, E. D. Taillard, and G. Agazzi. MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas In Optimization*. McGraw-Hill, 1999.

[82] L. M. Gambardella and M. Dorigo. Ant colony system hybridized with a new local search for sequential ordering problem. *INFORMS Journal on Computing.*, 12(3):237–255, 2000.

[83] M. R. Garey and D. S. Johnson. *Computers and intractability: A guide to the theory of NP-completeness.* W. H. Freeman, 1979.

[84] M. Gendreau, A. Hertz, and G. Laporte. A tabu search heuristic for the vehicle routing problem. report CRT-777, Centre de recherche sur les transports, Université de Montréal. *Management Science*, 1992.

[85] M. Gendreau, G. Laporte, and J.-Y. Potvin. Metaheuristics for the vehicle routing problem. In P. Toth and D. Vigo, editors, *The Vehicle Routing Problem, Volume 9 of SIAM Series on Discrete Mathematics and Applications.*, pages 129–154, 2001.

[86] B. Gillett and L. Miller. A heuristic algorithm for the vehicle dispatch problem. *Operations Research*, 22:340–349, 1974.

[87] F. Glover. Integer programming over a finite additive group. *SIAM Journal on Control.*, 7:213–231, 1969.

[88] F. Glover. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8:156–166, 1977.

[89] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13:533–549, 1986.

[90] F. Glover. Tabu search, part I. *ORSA Journal on Computing*, 1:190–206, 1989.

[91] F. Glover. Genetic algorithms and scatter search: Unsuspected potentials. *Statistics and Computing.*, 4(2):131–140, 1994.

[92] F. Glover. Scatter search and star-paths: Beyond the genetic metaphor. *OR Spectrum.*, 17(2-3):125–137, 1995.

[93] F. Glover. Tabu search and adaptive memory programing advances, applications and challenges. In R. Barr, R. Helgason, and J. Kennington., editors, *Interfaces in Computer Science and Operations Research.*, pages 1–75. Kluwer Academic Publishers, Boston, 1996.

[94] F. Glover. A template for scatter search and path relinking. In J. Hao, E. Lutton, E. Ronald, M. Schoenauer, and D. Snyers, editors, *Artificial Evolution*, volume 1363 of *Lecture Notes in Computer Science*, pages 13–54. Springer-Verlag, 1998.

[95] F. Glover. Scatter search and path relinking. Technical Report HCES-01-99, University of Mississippi, Hearin Center for Enterprise Science., 1999.

[96] F. Glover and M. Laguna. *Tabu Search.* Kluwer Academic Publishers: Boston, MA, 1997.

[97] F. Glover, M. Laguna, and R. Martí. Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 29(3):653–684, 2000.

[98] F. Glover, M. Laguna, and R. Martí. Scatter search and path relinking: Advances and applications. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics.*, pages 1–36. Kluwer Academic Publishers, Boston, 2003.

[99] F. Glover, M. Laguna, and R. Martí. Scatter search and path relinking: Foundations and advanced designs. In G. C. Onwubolu and B. V. Babu, editors, *New Optimization Techniques in Engineering*. Springer-Verlag, Heidelberg, 2004.

[100] D. E. Goldberg. *Genetic algorithms in search, optimization and machine learning.* Addison Wesley, Reading, MA, 1989.

[101] R. E. Gomory. On the relation between integer and non-integer solutions to linear programs. *Proceeding of the National Academy of Science.*, 53:260–265, 1965.

[102] R. E. Gomory. Some polyhedra related to combinatorial problems. *Linear Algebra and its Applications.*, 2:451–558, 1969.

[103] B. S. Gottfried and J. Weisman. *Introduction to Optimization Theory.* Prentice-Hall, Inc. New Jersey, 1973.

[104] M. Grötschel. Discrete mathematics in manufacturing. *Preprint SC92-3, ZIB*, 1992.

[105] L. Grover. Local search and the local structure of np-complete problems. *Operations Research Letters.*, 12:235–243, 1992.

[106] W. Gutjahr. A graph-based ant system and its convergence. *Future Gerneration Computer Systems*, 16(9):873 – 888, 2000.

[107] W. Gutjahr. Aco algorithms with guaranteed convergence to the optimal solution. *Information Processing Letters*, 82:145 – 153, 2002.

[108] W. Gutjahr. A generalized convergence result for the graph-based ant system metaheuristic. *Probability in the Engineering and Informational Sciences*, 17(4):545 – 569, 2003.

[109] K. Hamacher. Adaptation in stochastic tunneling global optimization of complex potential energy landscapes. *Europhys. Lett.*, 74(6):944–950, 2006.

[110] J. Hamiez and J. Hao. Scatter search for graph coloring. In P. Collet, C. Fonlupt, J.-K. Hao, E. Lutton, and M. Schoenauer, editors, *Artificial Evolution : 5th International Conference, Evolution Artificielle, EA 2001, France*, volume 2310 of *Lecture Notes in Computer Science*, pages 168–179. Springer Verlag, Berlin, Germany., 2002.

[111] P. Hansen, N. Mladenović, and D. Urošević. Variable neighborhood search for the maximum clique. *Discrete Applied Mathematics.*, 145(1):117–125, 2004.

[112] P. Hansen and N. Mladenovié. An introduction to variable neighborhood search. In S. Voss, S. Martello, I. Osman, and C. Roucairol, editors, *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization.*, pages 433–458. Dordrecht, Netherlands, Kluwer Academic Publishers, 1999.

[113] P. Hansen and N. Mladenovié. Variable neighborhood search: Principles and applications. *European Journal of Operational Research.*, 130:449–467, 2001.

[114] J. Hart and A. Shogan. Semi-greedy heuristics: An empirical study. *Operations Research Letters.*, 6:107–114, 1987.

[115] S. C. Ho and M. Gendreau. Path relinking for the vehicle routing problem. *Journal of Heuristics*, 12(1-2):55–72, 2006.

[116] J. Holland. *Adaption in Natural and Artificial Systems.* Univ. of Michigan Press: Ann Arbor. Reprinted in 1992 by MIT Press, Cambridge MA, 1975.

[117] C. Igel and M. Toussaint. On classes of functions for which no free lunch results hold. *Information Processing Letter.*, 86(6):317–321, 2003.

[118] L. Ingber. Adaptive simulated annealing (ASA): Lessons learned. *Control and Cybernetics - Special Issue on Simulated Annealing Applied to Combinatorial Optimization.*, 25(1):33–54, 1996.

[119] A. Jagota and L. A. Sanchis. Adaptive, restart, randomized greedy heuristics for maximum clique. *Journal of Heuristics.*, 7(6):565–585, 2001.

[120] C. Kanzow and A. Klug. An interior-point affine-scaling trust-region method for semismooth equations with box constraints. *Computational Optimization and Applications*, 37(3):329–353, 2007.

[121] V. S. Khaled AlSabti, Sanjay Ranka. An efficient space-partitioning based algorithm for the k-means clustering. In N. Zhong and L. Zhou, editors, *Proceedings of Methodologies for Knowledge Discovery and Data*

*Mining: Third Pacific-Asia Conference, PAKDD-99*, volume 1574 of *Lecture Notes in Computer Science*, pages 355–359,. Springer, 1999.

[122] G. W. Kinney. *A group theoretic approach to metaheuristic local search for partitioning problems*. PhD thesis, The University of Texas at Austin, 2005.

[123] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[124] B. Korte, editor. *Modern Applied Mathematics: Optimization and Operations Research*. North-Holland Publishing Company, 1982.

[125] N. Krasnogor. A tutorial on memetic algorithms. In *The 7th International Conference on Parallel Problem Solving from Nature - PPSN 2002, Spain.*, page 127, 2002.

[126] N. Krasnogor, D. Pelta, P. M. Lopez, P. Mocciola, and E. de la Canal. Genetic algorithms for the protein folding problem: A critical view. In E. Alpaydin and C. Fyfe, editors, *Proceedings of Engineering of Intelligent Systems, EIS'98.*, pages 353–360. ICSC Academic Press, 1998.

[127] P. J. M. V. Laarhoven, E. H. L. Aarts, and J. K. Lenstra. Job shop scheduling by simulated annealing. *Operations Research*, 40:113–125, 1992.

[128] M. Laguna. Scatter search. In P. M. Pardalos and M. G. C. Resende, editors, *Handbook of Applied Optimization*, pages 183–193. Oxford University Press, 2002.

[129] G. Laporte. The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(3):345–358, 1992.

[130] L. S. Lasdon. *Optimization Theory for Large System.* MacMillan Inc., London, 1970.

[131] E. Lawler, J. K. Lenstra, A. H. G. R. Kan, and D. B. Shmoys. *The Travelling Salesman Problem.* John Wiley and Sons, New York, NY, 1985.

[132] E. L. Lawler, J. K. Lenstra, A. H. G. R. Kan, and D. B. Shmoys. *The traveling salesman problem.* John Wiley, 1985.

[133] N. Lesh, M. Mitzenmacher, and S. Whitesides. A complete and effective move set for simplified protein folding. In *Proceedings of the seventh annual international conference on Computational molecular biology.*, pages 188–195. ACM Press, 2003.

[134] S. Lin. Computer solutions of the traveling salesman problem. *Bell System Technical Journal.*, 44:2245–2269, 1965.

[135] S. Lin and B. Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, 21:498–516, 1973.

[136] O. Martin, S. W. Otto, and E. W. Felten. Alarge-step markov chains for the traveling salesmanproblem. *Complex Systems*, 5(3):299–326, 1991.

[137] D. Merkle, M. Middendorf, and H. Schmeck. Ant colony optimization for resource-constrained project scheduling. *IEEE Transactions on Evolutionary Computation.*, 6(4):333–346, 2002.

[138] N. Metropolis, A. Rosenbluth, M. Rosenbluth, M. Teller, and E. Teller. Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1092, 1953.

[139] N. Metropolis, A. Rosenbluth, M. Rosenbluth, M. Teller, and E. Teller. *Equations of state calculations by fast computing machines.* Addison-Wesley, 2nd edition, 1989.

[140] Z. Michalewics. *Genetic Algorithms + Data Structures = Evolutionary Programs.* Springer: New York, 3rd edition, 1996.

[141] Z. Michalewicz and M. Michalewicz. Evolutionary computation techniques and their applications. In *Proceedings of the IEEE International Conference on Intelligence Processing System.*, pages 14–24. Institute of Electrical and Electronics Engineerings, Incorporated., 1997.

[142] W. Miehle. Link-length minimization in networks. *Operations Research*, 6:232–243, 1958.

[143] M. Mitchell. *An introduction to genetic algorithms.* MIT press, Cambridge, MA, 1998.

[144] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers and Operations Research.*, 24(11):1097–1100, 1997.

[145] J. Mockus, W. F. Eddy, A. Mockus, L. Mockus, and G. Reklaitis. *Bayesian Heuristic Approach to Discrete and Global Optimization: Algorithms, Visualization, Software, and Applications (Nonconvex Optimization and Its Applications).* Dordrecht, Kluwer Academic Publishers, 1997.

[146] P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Caltech Concurrent Computation Program. Report 826, California Institute of Technology, Pasadena, California, USA., 1989.

[147] P. Moscato. Memetic algorithms: A short introduction. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas In Optimization*, pages 219–234. McGraw-Hill, 1999.

[148] H. Mühlenbein and G. Paass. From recombination of genes to the estimation of distributions. In H. M. Voigt, W. Ebeling, I. Rechenberg, and H. P. Schwefel, editors, *Proceedings of the 4th Conference on Parallel Problem Solving from Nature - PPSN IV*, volume 1411 of *Lecture Notes in Computer Science*, pages 178–187. Springer, 1996.

[149] S. Mulder and D. W. II. Million city traveling salesman problem solution by divide and conquer clustering with adaptive resonance neural networks. *Neural Networks*, 16(5-6):827–832, 2003.

[150] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization.* John Wiley and Sons, New Jork, 1999.

[151] A. S. Nemirovsky and D. B. Yudin, editors. *Problem Complexity and Method Efficiency in Optimization.* John Wiley and Sons translated by E. R. Dawson, 1983.

[152] P. L. nga and J. A. Lozano, editors. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation.* Kluwer Academic Publishers, Boston, MA, 2002.

[153] E. Nowicki and C. Smutnicki. A fast taboo search algorithm for the job-shop scheduling. *Management Science*, 42(5):797–813, 1996.

[154] W. Orchard-Hays. *Advanced Linear-Programming Computing Techniques.* McGraw-Hill Book Company, New York, 1968.

[155] I. Osman. Metastrategy simulated annealing and tabu search algorithms employing a generalised savings criterion. *Annals of Operations Research*, 41(1–4):421–451, 1993.

[156] I. H. Osman and G. Laporte. Metaheuristics: A bibliography. *Annals of Operations Research*, 63:513–623, 1996.

[157] F. M. P. Frana, A. Mendes and P. Moscato. Memetic algorithms applied to the single machine and parallel machine scheduling problems. In *Anais da Primeira Oficina de Planejamento e Controle da Produo em Sistemas de Manufatura*, April 1999. projeto temtico FAPESP 97/13930-1, Campinas, SP.

[158] S.-M. Pan and K.-S. Cheng. Evolution-based tabu search approach to automatic clustering. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 37(5):827–838, 2007.

[159] C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley Inc., 1994.

[160] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization - Algorithms and Complexity.* Dover Publications, Inc., New York, 2nd edition, 1998.

[161] P. Pardalos, L. Pitsoulis, and M. Resende. A parallel GRASP implementation for the quadratic assignment problem. In A. Ferreira and J. Rolim, editors, *Parallel Algorithms for Irregularly StructuredProblems - Irregular'94.* Kluwer Academic Publishers, 1995.

[162] P. Pardalos, T. Qian, and M. Resende. A greedy randomized adaptive search procedure for feedback vertex set. *Journal of Combinatorial Optimization.*, 2(4):399–412, 1998.

[163] A. L. Patton, W. F. P. III, and E. D. Goodman. A standard GA approach to native protein conformation prediction. In *ICGA*, pages 574–581, 1995.

[164] J. P. Pedroso. Simple metaheuristics using the simplex algorithm for non-linear programming. In T. Sttzle, M. Birattari, and H. H. Hoos, editors, *SLS*, volume 4638 of *Lecture Notes in Computer Science*, pages 217–221. Springer, 2007.

[165] M. Pilat and T. White. Using genetic algorithms to optimize ACS-TSP,. In M. Dorigo, G. D. Caro, and M. Samples, editors, *ANTS 2002, From Ant Colonies To Artificial Ants: The Third International Workshop on Ants Algorithms*, pages 282–287. Springer, 2002.

[166] J. R. Quinlan. Combining instance-based and model-based learning. In *Proceedings of the Tenth International Conference on Machine Learning.*, pages 236–243. Morgan Kaufmann, Amherst, Massachusetts, 1993.

[167] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* PhD thesis, Fromman-Holzboog, Stuttgart, 1973.

[168] C. R. Reeves. *Modern Heuristic Techniques for Combinatorial Problems.* Blackwell Scientific Publishing, Oxford, England, 1993.

[169] C. R. Reeves and J. E. Rowe. *Genetic Algorithms: Principles and Perspectives. A Guide to GA Theory.* Kluwer Academic Publishers, Boston (USA), 2002.

[170] M. Reimann, K. Doerner, and R. Hartl. D-ants: Savings based ants divide and conquer the vehicle routing problem. *Computers and Operations Research*, 31:563–591, 2004.

[171] D. H. P. Rene Cori, Daniel Lascar. *Mathematical Logic: A Course with Exercises Part I: Propositional Calculus, Boolean Algebras, Predicate Calculus, Completeness Theorems* . Oxford University Press, USA, 2000.

[172] M. Resende and C. Ribeiro. Grasp with path-relinking: Recent advances and applications. In K. Nonobe and M. Yagiura, editors, *Metaheuristics: Progress as Real Problem Solvers*. Kluwer Academic Publishers, 2005.

[173] M. G. C. Resende, L. S. Pitsoulis, and P. M. Pardalos. Fortran subroutines for computing approximate solutions of weighted MAX-SAT problems using GRASP. *Discrete Applied Mathematics*, 100(1–2):95–113, 2000.

[174] C. C. Ribeiro and M. C. Souza. Variable neighborhood search for the degree-constrained minimum spanning tree problem. *Discrete Applied Mathematics.*, 118(1-2):43–54, 2002.

[175] A. J. Robertson. A set of greedy randomized adaptive local search procedure GRASP implementations for the multidimensional assignment problem. *Computational Optimization and Applications.*, 19(2):145–164, 2001.

[176] P. Salamon, P. Sibani, and R. Frost. *Facts, Conjectures, and Improvements for Simulated Annealing*. SIAM, 2003.

[177] H. S. Sánchez and J. F. Solís. A method to establish the cooling scheme in simulated annealing like algorithms. In *ICCSA (3)*, pages 755–763, 2004.

[178] J. G. Shanthikumar and Y. B. Wu. Decomposition approaches in permutation scheduling with application to the *m*-machine flow shop scheduling problems. *European Journal of Operations Research.*, 19:125–141, 1985.

[179] Y. Shen, S. Kiatsupaibul, Z. B. Zabinsky, and R. L. Smith. An analytically derived cooling schedule for simulated annealing. *Journal of Global Optimization*, 38(3):333–365, 2007.

[180] A. Shmygelska, R. A. Hernández, and H. H. Hoos. An ant colony optimization algorithm for the 2D HP protein folding problem. In *Ant Algorithms*, pages 40–53, 2002.

[181] A. Sinha and D. E. Goldberg. A survey of hybrid genetic and evolutionary algorithms. Technical Report 2003004, IlliGAL, Urbana : University of Illinois at UrbanaChampaign, General Engineering Department, Jan 2003.

[182] J. Smith. On replacement strategies in steady state evolutionary algorithms. *Evolutionary Computation*, 15(1):29–59, 2007.

[183] W. M. Spears, K. A. D. Jong, T. Bück, D. B. Fogel, and H. D. Garis. An overview of evolutionary computation. In P. B. Brazdil, editor, *Proceedings of the European Conference on Machine Learning ECLM-93*, volume 667, pages 288–289. Springer Verlag, Vienna, Austria, 1993.

[184] P. Stadler. Spectral landscape theory. In J. Crutchfield and P. Schuster, editors, *Evolutionary Dynamics – Exploring the Interplay of Selection, Neutrality, Accident and Function*. Oxford University Press, New York, 1999.

[185] P. F. Stadler. Landscapes and their correlation functions. *Journal of Mathematical Chemistry.*, 20:1–45, 1996.

[186] T. Stützle, editor. *Local Search Algorithms for Combinatorial Optimization Problems: Analysis, Improvements, and New Applications*. vol 220 of DISKI. Sankt Augustin, Germany, Infix, 1999.

[187] T. Stützle. *Local Search Algorithms for Combinatorial Problems - Analysis, Algorithms and New Applications*. Dissertations in Artificial Intelligence-Infix, Sankt Augustin, Germany, 1999.

[188] T. Stützle and M. Dorigo. ACO algorithms for the quadratic assignment problem. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas In Optimization*. McGraw-Hill, 1999.

[189] T. Stützle and M. Dorigo. A short convergence proof for a class of ACO algorithms. *IEEE Transactions on Evolutionary Computation*, 6(4):358–365, 2002.

[190] T. Stützle and H. Hoos. The MAX-MIN ant system and local search for the traveling salesman problem. In T. Bäck, Z. Michalewicz, and X. Yao, editors, *Proceedings of the 4th International Conference on Evolutionary Computation (ICEC'97)*, pages 308–313. IEEE Press, 1997.

[191] T. Stützle and H. Hoos. MAX-MIN ant system and local search for combinatorial optimization problems. In S. Voss, S. Martello, I. Osman, and C. Roucairol, editors, *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization.*, pages 137–154. Dordrecht, Netherlands, Kluwer Academic Publishers, 1999.

[192] C. S. Sung, Y. H. Kim, and S. H. Yoon. A problem reduction and decomposition approach for scheduling for a flowshop of batch processing machines. *European Journal of Operational Research.*, 121:179–192, 2000.

[193] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, 1998.

[194] E. D. Taillard. Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 17:443–455, 1991.

[195] E. D. Taillard. Parallel iterative search methods for vehicle routing problems. *Networks*, 23:661–73, 1993.

[196] E. D. Taillard and S. Voss. POPMUSIC–partial optimization metaheuristic under special intensification conditions. In C. C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics.*, pages 613–629. Kluwer Academic Publisher, 2002.

[197] R. Unger and J. Moult. Genetic algorithms for protein folding simulations. *Biochemistry*, 231(1):75–81, 1993.

[198] A. Uwe and D. Dipankar. Chapter 13: Artificial immune systems tutorial. In E. Burke and G. Kendall, editors, *Search Methodologies - Introductory Tutorials in Optimization and Decision Support Techniques*, pages 375–399. Springer, 2006.

[199] V. Valls, S. Quintanilla, and F. Ballestín. A population based approach to the resource constrained project scheduling. Technical Report TR06-2001, Departamento de Estadística e Investigación Operativa, Facultad de Matemáticas, Universitat de València, Spain, 2001.

[200] V. Černý. A thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45:41–51, 1985.

[201] M. D. Vose. *The simple genetic algorithm: foundations and theory.* MIT press, Cambridge, MA, 1999.

[202] S. Voß, S. Martello, I. H. Osman, and C. Roucairol. *Meta-Heuristics - Advances and Trends in Local Search Paradigms for Optimization.* Kluwer Academic Publisher, Dordrecht, The Netherlands, 1999.

[203] C. Voudouris and E. Tsang. Guided local search. Technical Report CSM-247, Department of Computer Science, University of Essex., 1995.

[204] D. Whitley and J. Watson. Complexity theory and the no free lunch theorem. In E. Burke and G. Kendall, editors, *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, pages 317–339. Springer, 2006.

[205] D. H. Wolpert and W. G. Macready. No free lunch theorems for search. Technical Report SFI-TR-95-02-010, Santa Fe Institute., 1995.

[206] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation.*, 1(1):67–82, 1997.

[207] L. A. Wolsey. *Mixed Integer Programming: Discretization and the Group Theoretic Approach*. PhD thesis, Massachusetts Institute of Technology, 1969.

[208] M. Yagiura and T. Ibaraki. On metaheuristic algorithms for combinatorial optimization problems. *Systems and Computers in Japan.*, 32:33–55, 2001.